HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Andreas Schmidt

Konzeptionierung und Erstellung einer Betriebs- und Konfigurationssoftware für einen CAN-Bus-Datenlogger

Mittweida, 2016

Fakultät Angewandte Computer- und Biowissenschaften

DIPLOMARBEIT

Konzeptionierung und Erstellung einer Betriebs- und Konfigurationssoftware für einen CAN-Bus-Datenlogger

Autor:
Herr

Andreas Schmidt

Studiengang:
Informationstechnik

Seminargruppe:
KI11w1-F

Erstprüfer:
Prof. Dr.-Ing. Olaf Hagenbruch

Zweitprüfer:
Dipl.-Ing (FH) Martin Jakob

Einreichung:
Mittweida, 30.03.2016

Verteidigung/Bewertung:
Mittweida, 2016

Bibliografische Beschreibung:

Schmidt, Andreas:
Konzeptionierung und Erstellung einer Betriebs- und Konfigurationssoftware für einen CAN-Bus-Datenlogger - 2016 - VII, 59, 1 S.
Mittweida, Hochschule Mittweida, Fakultät Angewandte Computer- und Biowissenschaften, Diplomarbeit, 2016

Referat:

Die vorliegende Diplomarbeit befasst sich mit der Erstellung eines Konzeptes für eine Betriebs- und Konfigurationssoftware eines CAN-Bus-Datenloggers sowie mit der anschließenden Implementierung dieses Konzeptes. Basierend auf einer Analyse bestehender Datenlogger werden die notwendigen Anforderungen ermittelt. Diese Anforderungen stellen die Grundlage für das Konzept der Betriebs- und Konfigurationssoftware dar. Im Anschluss an die Implementierung erfolgt der Nachweis über die Umsetzung der vorab definierten Anforderungen mittels Verifikation.

Inhalt

Inhalt ..... **I**

Abbildungsverzeichnis ..... **III**

Tabellenverzeichnis ..... **V**

Abkürzungsverzeichnis ..... **VI**

1 Einleitung **1**

1.1 Motivation *1*

1.2 b.cube *2*

2 CAN-Bus-Datenlogger **3**

3 Anforderungen **6**

4 Konzeptioneller Entwurf **8**

4.1 Randbedingungen *8*

4.1.1 Maximale Verarbeitungszeit je CAN-Botschaft 8

4.1.2 Vorgaben Hardware und Programmiersprache 9

4.2 Übergeordnete Konzeptthemen *10*

4.2.1 Speicherformat 10

4.2.2 Speichermedium 13

4.2.3 Schnittstellendefinition 14

4.3 Betriebssoftware-Konzept *17*

4.3.1 USB-Stick-Anbindung 17

4.3.2 Anbindung, Konfiguration und Auslesen von CAN-Schnittstellen 17

4.3.3 Empfang und Umsetzung von Kommandos der Konfigurationssoftware 19

4.3.4 Überprüfung von CAN-Botschaften auf Trigger-Bedingungen 20

4.3.5 Pufferspeicher für Vorlauf 24

4.3.6 Starten und Stoppen der Aufzeichnung 26

4.3.7 Verarbeitung und Speicherung von CAN-Botschaften 28

4.3.8 Speichern der Konfiguration in einem nichtflüchtigen Speicher 30

4.3.9 Struktur der Betriebssoftware 31

4.4 Konfigurationssoftware-Konzept *32*

4.4.1 Auswahl Programmiersprache und Grafik-Framework 32

4.4.2 Ermittlung der Use-Cases ... 33
4.4.3 Grundkonzept der Benutzeroberfläche ... 34
4.4.4 Umsetzung der Use-Cases ... 36

5 Implementierung ... 40

5.1 Betriebssoftware ... 40

5.2 Konfigurationssoftware ... 43
5.2.1 MainWindow ... 44
5.2.2 App.xaml ... 45
5.2.3 ContentPage ... 45
5.2.4 WindowCOM ... 46
5.2.5 PageCANInterfaces ... 47
5.2.6 PageDatalogger ... 47
5.2.7 Konfiguration des Dataloggers ... 50
5.2.8 Import und Export von LogDateien ... 50

6 Softwareverifikation ... 52

6.1 Testumgebung ... 52

6.2 Testfalldefinition ... 53

6.3 Testergebnisse ... 57

7 Fazit und Ausblick ... 59

Literatur ... 60

Anlagen ... 64

Anlagen, Teil 1 ... A-1

Selbstständigkeitserklärung

Abbildungsverzeichnis

Abbildung 1 - Überblick b.cube [3] 2

Abbildung 2 - Signale in CAN-Botschaften 4

Abbildung 3 – Signalzuordnung einer CAN-Botschaft 5

Abbildung 4 - EA LPC4088 QuickStart Board [8] 9

Abbildung 5 - Inhalt einer ASC-Datei 10

Abbildung 6 - ASC CAN-Datensatz 10

Abbildung 7- ASC Trigger-Kennzeichnung 11

Abbildung 8 - Hardwaresicht Konfigurationsschnittstelle 14

Abbildung 9 - Kommunikationsablauf 16

Abbildung 10 - Verarbeitung von empfangenen Konfigurationskommandos 20

Abbildung 11 - Konzept der Trigger-Klasse der Betriebssoftware 21

Abbildung 12 - Ablauf Triggerprüfung 23

Abbildung 13 - Vorlaufpuffer 25

Abbildung 14 - Einfügen von Datensätzen in den Vorlaufpuffer 26

Abbildung 15- Aktivitätsdiagramm „Aufzeichnung stoppen“ 27

Abbildung 16 - Datenverarbeitung im Permanent-Modus 28

Abbildung 17 - Datenverarbeitung im Trigger-Modus 30

Abbildung 18 - Konzept Betriebssoftware 31

Abbildung 19 - Anwendungsfalldiagramm „Konfigurationssoftware“ 33

Abbildung 20 - Konzeptskizze Benutzeroberfläche der Konfigurationssoftware 34

Abbildung 21 - Konzept Konfigurationssoftware Datenklassen 36

Abbildung 22 - Senden von Konfigurationskommandos 38

Abbildung 23 - Module der Betriebssoftware 40

Abbildung 24 - Klassendiagramm Betriebssoftware 41

Abbildung 25 - Komponenten der Konfigurationssoftware 44

Abbildung 26 - MainWindow 44

Abbildung 27 - Übersicht der von ContentPage abgeleiteten Klassen 45

Abbildung 28 - WindowCOM 46

Abbildung 29 - Ablaufdiagramm getDevices 46

Abbildung 30 - PageCANInterfaces 47

Abbildung 31 - Benutzeroberfläche Interface- und Betriebsmodus-Konfiguration 48

Abbildung 32 - Benutzeroberfläche Trigger-Konfiguration 49

Abbildung 33 - WindowTrigger 49

Abbildung 34 - Klassenbeziehungen der Konfigurationssoftware 50

Abbildung 35 - PageLogView 51

Abbildung 36 - Testaufbau 52

Tabellenverzeichnis

Tabelle 1 - Beispiel CAN-Matrix 5

Tabelle 2 - Datensatz im binären Format 12

Tabelle 3 - Bewertung Speichermedien 13

Tabelle 4- Kommandotabelle 15

Tabelle 5 - Attribute eines CANMessage-Objektes 18

Tabelle 6 – Datenbytes der zu überprüfenden CAN-Botschaft 22

Tabelle 7 – Verschiebung des Referenzwert für Triggerprüfung 22

Tabelle 8 - Erstellung der Trigger-Bitmaske 22

Tabelle 9 - Triggerprüfung maskieren der Datenbits 23

Tabelle 10 - Zuordnung von Uses-Cases zu Menüpunkten 35

Tabelle 11 - Testfall T01 53

Tabelle 12 - Testfall T02 53

Tabelle 13 - Testfall T03 54

Tabelle 14 - Testfall T04 54

Tabelle 15 - Testfall T05 55

Tabelle 16 - Testfall T06 55

Tabelle 17 - Testfall T07 56

Tabelle 18 - Testfall T08 56

Tabelle 19 - Testfall T09 57

Tabelle 20 - Auswertung der Testfälle 57

Abkürzungsverzeichnis

ACK	Acknowledge
API	Application programming interface
App	Application software
ASC	ASCII Frame Logging File
BFL	Binary Logging Format
CAN	Controller Area Network
CMH	Compiled HTML Help
Deque	Double-Ended Queue
DLC	Data-Length-Code
EOF	End of Frame
HTML	Hypertext Markup Language
ID	Identifier
IDE	Integrated Development Environment
ISR	Interrupt Service Routine
LAN	Local Area Network
LIN	Local Interconnect Network
QSPI	Queued Serial Peripheral Interface
RAM	Random-Access Memory
RTR	Remote Transmission Request
Rx	Receiver
Tx	Transmitter
USB	Universal Serial Bus
SD-Karte	Secure Digital Memory Card
SOF	Start of Frame
SRAM	Static Random-Access Memory

WLAN	Wireless Local Area Network
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
z.B.	zum Beispiel

1 Einleitung

Ziel dieser Diplomarbeit ist die Konzeptionierung und Implementierung einer Betriebs- und Konfigurationssoftware eines CAN-Bus-Datenloggers. Nach diesem einleitenden Kapitel wird zunächst eine, Analyse zum Funktionsumfang der am Markt vorhandenen CAN-Bus-Datenlogger durchgeführt. In Kapitel 3 werden auf Basis der aus der Analyse gewonnen Erkenntnisse die genauen Anforderungen an die Betriebs- und Konfigurationssoftware definiert. Diese Anforderungen stellen die Grundlage für den konzeptionellen Entwurf des vierten Kapitels dar. Kapitel 5 beschreibt die Implementierung des erarbeiteten Konzeptes. Mittels der Softwareverifikation in Kapitel 6 wird der Nachweis erbracht, dass die implementierte Betriebs- und Konfigurationssoftware die definierten Anforderungen erfüllt.

1.1 Motivation

Die Steuerung von Funktionen im Fahrzeug hat sich in den letzten Jahrzehnten stark verändert. Die direkte Ansteuerung von Aktoren im Fahrzeug über Schalter, mittels direkter Verdrahtung, wurde durch ein komplexes System von untereinander vernetzten Steuergeräten ersetzt. Die Vernetzung der Steuergeräte erfolgt heutzutage hauptsächliche über den CAN-Bus (Controller Area Network). In aktuellen Fahrzeugen findet man bis zu 100 untereinander vernetzte Steuergeräte [1]. Die Steuergeräte werden nicht alle am gleichen Bus-System betrieben, sondern anhand ihrer Funktion in verschiedene Busse aufgeteilt. Verbreitet ist eine Aufteilung in Antriebsbus, Komfortbus und Infotainmentbus. Da häufig auch eine Kommunikation unter Steuergeräten verschiedener Busse erfolgt, muss ein Routing der busübergreifenden Nachrichten erfolgen. Dieses Routing wird von einem speziellen Steuergerät, dem Gateway, realisiert. Das Gateway ist an alle CAN-Busse des Fahrzeuges angebunden und kann bei Bedarf Nachrichten auf andere Busse weiterleiten. Aufgrund der hohen Datenabhängigkeit der Steuergeräte voneinander, kommt es während der Entwicklung oft vor, dass eine Funktion noch nicht getestet werden kann, da eine Botschaft von einem noch nicht vorhandenen Steuergerät fehlt. Dieses Problem kann durch Simulation der fehlenden Botschaften umgangen werden. Dieses Verfahren nennt man Restbussimulation. Aufgrund der hohen Komplexität der Abhängigkeit der Steuergeräte untereinander, ist es häufig notwendig, die Nachrichten der CAN-Busse zur späteren Fehleranalyse aufzuzeichnen. Man spricht hier von Datenlogging [2]. Die Funktionalitäten zum Datenlogging und zur Restbussimulation werden über zusätzliche Hardware realisiert. Die Firma Bertrandt Ingenieurbüro GmbH Tappenbeck hat sich das Ziel gesetzt, ein System zu entwickeln, welches fahrzeugherstellerunabhängig einsetzbar ist und die Funktionen Routing, Restbussimulation und Datenlogging in einem Gerät vereint. Das Gesamtsystem aus Software und Hardware erhält den Produktnamen „b.cube“. Die Entwicklung der dafür erforderlichen Module erfolgt in mehreren studentischen Arbeiten. Diese Diplomarbeit beschäftigt

sich mit der Konzeptionierung und Erstellung der Betriebs- und Konfigurationssoftware des CAN-Bus-Datenloggers dieses Produktes.

1.2 b.cube

Ziel des b.cube-Projektes ist die Realisierung eines Systems, mit welchem eine Vielzahl von unterschiedlichen Anwendungsfällen in den Bereichen der Fahrzeugvernetzung, Analyse und Test flexible abgedeckt werden können. Die Abbildung 1 gibt einen Überblick über das Grundkonzept und die Funktionalität des Systems b.cube.

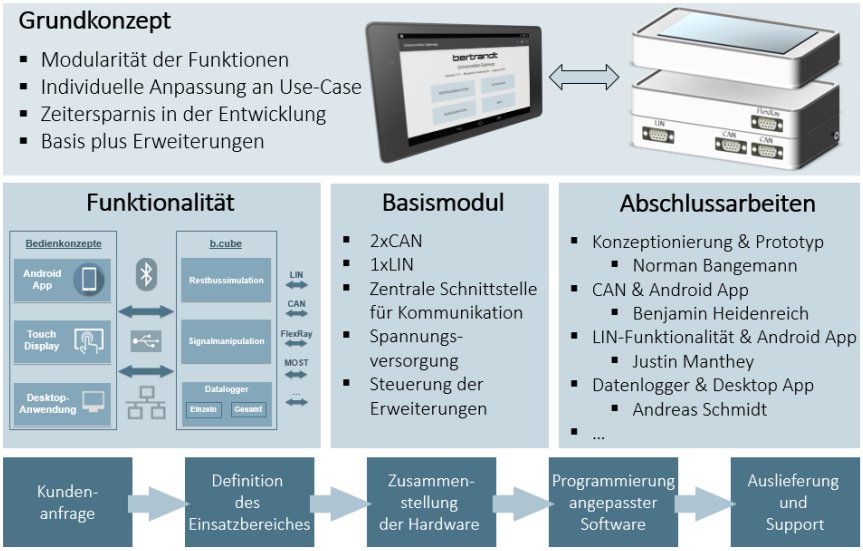

Abbildung 1 - Überblick b.cube [3]

Das System b.cube wird modular aufgebaut und kann bei Bedarf um zusätzliche Schnittstellen und Funktionen mittels Steckmodulen erweitert werden [3].

Das Basismodul umfasst zwei CAN-Schnittstellen, eine LIN (Local Interconnect Network)-Schnittstelle und eine USB (Universal Serial Bus)-Schnittstelle. Außerdem enthält es die Komponenten zur Energieversorgung und das Kommunikationsmanagement. Das Kommunikationsmanagement realisiert die systeminterne Kommunikation zu den Erweiterungsmodulen sowie die Schnittstellen zu den verschieden Bedienoberflächen [3].

Die einzelnen Funktionen des Systems lassen sich mit einer Windowsanwendung über USB konfigurieren und steuern. Alternativ kann die Bedienung über ein direkt an das System angebundenes Touch-Display erfolgen, welches als Zusatzmodul erhältlich ist. Optional ist eine Steuerung und Konfiguration per Android mittels Bluetooth möglich [3].

Das System b.cube bietet für jede verbaute Bus-Schnittstelle die Funktionen des Datalogging, der Restbussimulation sowie der Signalmanipulation und des Routings. Für weitere Schnittstellen, wie zum Beispiel einen Analog-Eingang, können weitere Funktionalitäten adaptiert werden [3].

2 CAN-Bus-Datenlogger

Dieses Kapitel gibt einen Überblick über den Funktionsumfang der derzeit auf dem Markt erhältlichen CAN-Bus-Datenlogger. In Rahmen dieser Diplomarbeit erfolgt bei der Marktanalyse eine Beschränkung auf Produkte, welche bei der Bertrandt Ingenieurbüro GmbH Tappenbeck oder im Kundenumfeld der Firma eingesetzt werden. Für das Logging von CAN-Bussen werden Geräte der Firmen G.i.N. mbH (Gesellschaft für industrielle Netzwerke), Telemotiv AG und Vector Informatik GmbH verwendet. Jeder der Hersteller bietet mehrere Datenlogger an, die zusätzlich zu CAN-Bus noch weitere Bus-Systeme aufzeichnen können, z.B. LIN, MOST (Media Oriented Systems Transport) und Ethernet. Außerdem verfügen die von den drei Herstellern angebotenen Produkte zum Teil über Funktionalitäten zum Routen und zur Simulation von CAN-Bus-Botschaften.

Die Analyse der ausgewählten Datenlogger beschränkt sich auf die Funktionen zum Aufzeichnen von CAN-Bus-Botschaften. Basis für die Analyse sind die auf den Herstellerseiten verfügbaren Produktinformationen und Handbücher [4][5][6].

In Abhängigkeit von Funktionsumfang und Preisklasse der Datenlogger können die aufgezeichneten Daten auf SD-Karten (Secure Digital Memory Card), USB-Sticks und/oder internen Festplatten gespeichert werden. Das Speicherformat ist herstellerspezifisch, allerdings bieten alle Datenlogger die Möglichkeit die Daten im Measurement Data Format (MDF) oder im ASCII-Format(ASC) der Firma Vector Informatik aufzuzeichnen oder in diese Formate zu konvertieren.

Die Konfiguration der Datenlogger erfolgt über die herstellereigene Konfigurationssoftware. Die Software ermöglichen unteranderem die Einstellung der Bitraten der CAN-Schnittstellen, außerdem lassen sich einzelne Schnittstellen durch die Konfigurationssoftware deaktivieren. Die Übertragung der Konfiguration erfolgt, je nach Modell und Hersteller, per SD-Karte, USB-Verbindung, LAN (Local Area Network) oder WLAN (Wireless Local Area Network). Über diese Schnittstellen lassen sich bei einigen Modellen auch die aufgezeichneten Daten auf den PC übertragen. Bei preiswerten Modellen muss zur Übertragung der Daten das Speichermedium direkt an den PC angeschlossen werden.

Der Start der Aufzeichnung erfolgt bei den Datenloggern der drei Hersteller manuell oder automatisch bei Empfang einer CAN-Botschaft. Außerdem unterstützen die Datenlogger einen Sleep-Mode, in welchem die Aufzeichnung automatisch beendet, wenn über einen bestimmten Zeitraum keine CAN-Botschaft empfangen wurde.

Bei den Datenloggern lassen sich Bedingung (Trigger) konfigurieren, bei deren Auftreten die Aufzeichnung gestartet oder gestoppt wird. Alternativ zu der Verwendung von Triggern

als Start- Stoppbedingungen für Aufzeichnungen, können die Datenlogger auch so konfiguriert werden, dass bei Auftreten eines Triggers alle Botschaften innerhalb einer definierten Vor- und Nachlaufzeit aufgezeichnet werden. Innerhalb der Nachlaufzeit können keine neuen Trigger ausgelöst werden. Als Trigger kann das Auftreten bestimmter CAN-Botschaften, das Auftreten bestimmter Signalwerte innerhalb einer CAN-Botschaft oder das betätigen eines Tasters konfiguriert werden. Bestimmte CAN-Botschaften können entweder Botschaften mit einem bestimmten CAN-Identifier oder Error-Frames sein. Der Begriff Signal bezeichnet in diesem Fall Teile des Datenfeldes eines CAN-Datenframes.

Im Datenfeld eines Daten-Frames werden häufig mehre Werte übertragen. Zum Beispiel kann im Datenfeld einer Motor-Botschaft die *Motortemperatur*, die *Drehzahl* sowie die aktuelle *Geschwindigkeit* enthalten sein (Abbildung 2).

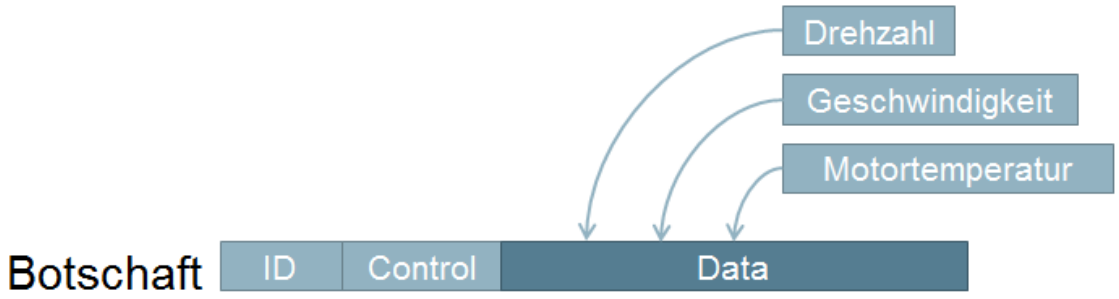

Abbildung 2 - Signale in CAN-Botschaften

Die einzelnen Signale *Drezahl*, *Motortemperatur* und *Geschwindigkeit* werden innerhalb des Datenfeldes übertragen werden. Jede Botschaftt kann somit mehre Signale enthalten. Damit die verschiedenen Steuergeräte die Signale verarbeiten können müssen sowohl der Sender als auch der Empfänger die genaue Position innerhalb des Signals im Datenfeld einer CAN-Botschaft kennen. Die einer Botschaft zugeordneten Signale und deren Positionen innerhalb der Botschaften können der fahrzeugspezifischen CAN-Matrix, oft auch Datenbasis genannt, entnommen werden. Darüber hinaus enthält eine Datenbasis die benötigten Faktoren zur Umrechnung des Signalrohwertes in den physikalischen Wert. Die Beschreibung des Signals innerhalb des Datafeldes erfolgt über die Angabe des Start-Bytes, des Start-Bits und der Länge des Signals in Bit. Tabelle 1 zeigt eine beispielhafte CAN-Matrix [7].

Tabelle 1 - Beispiel CAN-Matrix

Botschaft	ID	Signal	Startbyte	Startbit	Länge	Faktor	Einheit
Motor_01	0x100	Drehzahl	0	0	14	1	u/min
Motor_01	0x100	Geschwindigkeit	1	6	20	0,01	km/h
Motor_01	0x100	Temperatur	4	2	12	0,1	°C

Bei der Anwendung der CAN-Matrix ist zu beachten, dass sowohl die Bytes wie auch deren Bits von 0 bis 7 durchnummeriert werden. Die Abbildung 3 zeigt beispielhaft fünf Datenbytes einer CAN-Botschaft. In der Abbildung sind die drei Signale *Drehzahl*, *Geschwindigkeit* und *Temperatur* dargestellt.

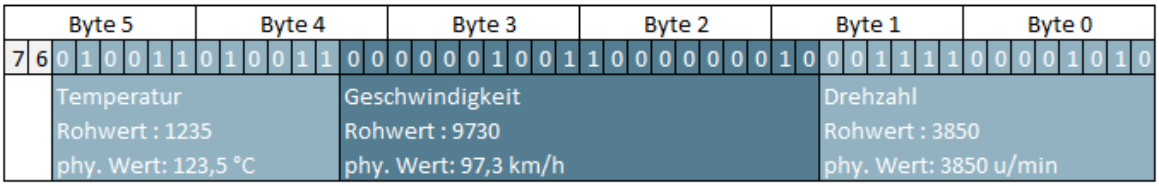

Abbildung 3 – Signalzuordnung einer CAN-Botschaft

Die Rohwerte und die physikalischen Werte werden mittels der CAN-Matrix aus Tabelle 1 ermittelt.

3 Anforderungen

Im Nachfolgenden werden die Anforderungen an die zu entwickelnde Betriebs- und Konfigurationssoftware des Datenloggers beschrieben. Die Anforderungen wurden in Zusammenarbeit mit den zukünftigen Anwendern definiert. Ergänzend fließen die Erkenntnisse aus der Recherche zum Funktionsumfang der im Kundenumfeld eingesetzten Datalogger ein (Kapitel 2). Zu jeder Anforderung gehören Akzeptanzkriterien in Form von einem oder mehreren Testfällen, die entsprechenden Testfälle sind in Kapitel 6.2 zu finden. Damit später eine Zuordnung zwischen Anforderung und Testfällen erfolgen kann werden die Anforderungen von A01 – A08 durchnummeriert.

A01 Betriebsarten

Der Datenlogger soll zwei verschiedene Betriebsarten unterstützen. Im Permanent-Modus soll eine durchgängige Aufzeichnung aller empfangenen CAN-Botschaften erfolgen. Ist der Trigger-Modus eines Datenloggers aktiv, soll die Aufzeichnung nur bei Auftreten einer vom Anwender definierten Bedingung(Trigger), mit einem Vor- und Nachlauf von zwei Sekunden, erfolgen.

A02 CAN-Schnittstellen

Die Bitrate der einzelnen CAN-Schnittstellen soll über die Konfigurationssoftware einstellbar sein. Außerdem sollen die Schnittstellen durch den Anwender deaktiviert werden können, sodass keine Aufzeichnung der an diesen Schnittstellen empfangenen CAN-Botschaften erfolgt.

A03 Trigger-Bedingungen

Der Anwender soll die Möglichkeit haben eine oder mehrere Bedingungen zum Auslösen eines Triggers zu konfigurieren. Als Bedingung sind vom Anwender ein CAN-Signal (Kapitel 2), ein Vergleichswert und ein Vergleichsoperator anzugeben. Ein Trigger ist auszulösen, wenn der Wert des CAN-Signals die Bedingung des Vergleichsoperators in Bezug zum Vergleichswert erfüllt. Als Vergleichsoperatoren sind gleich, ungleich, größer und kleiner vorzusehen.

A04 Manuelles Auslösen von Triggern

Alternativ, zum Auslösen von Triggern durch CAN-Signale, soll der Anwender die Möglichkeit haben, einen Trigger manuell mittels Betätigung eines Tasters auszulösen. Auch bei manuell ausgelösten Triggern ist ein Vor- und Nachlauf von zwei Sekunden zu realisieren.

A05 Konfigurationssoftware

Der Datenlogger soll über eine Windows-Desktopanwendung konfiguriert werden. Zusätzlich ist eine Schnittstelle zur Konfiguration über Bluetooth mittels Android-Applikation vorzusehen. Hier wird lediglich eine Schnittstelle vorgesehen. Die Android-Applikation ist nicht Bestandteil dieser Diplomarbeit.

A06 Starten und Stoppen der Aufzeichnung

Die Aufzeichnung von Daten durch den Datenlogger soll durch die Konfigurationssoftware gestartet und gestoppt werden können. Außerdem soll der Datenlogger über einen Modus verfügen, in dem die Aufzeichnung automatisch beim Empfang einer CAN-Botschaft gestartet wird. Im Automatikmodus ist die Aufzeichnung zu beenden, wenn innerhalb einer Minute keine CAN-Botschaften vom Datenlogger empfangen wurden.

A07 Speichermedium und Speicherformat

Als Speichermedium für die vom Datenlogger aufgezeichneten Daten ist ein Wechseldatenträger, eine SD-Karte oder ein USB-Stick zu verwenden. Die Daten sind nach Möglichkeit im Binary Logging Format (BFL) oder als ASCII Logging Files (ASC) zu speichern. Sollte dieses nicht möglich sein, ist alternativ ein Konverter für die Konvertierung der Logdatei in ein entsprechendes Format in der Konfigurationssoftware zu implementieren. Die vom Datenlogger aufgezeichneten Daten sollen in die Software CANoe importiert werden können.

A08 Laden und Speichern von Konfigurationen

Die Konfigurationssoftware soll dem Anwender die Möglichkeit bieten, Konfigurationen für den Datenlogger zu speichern und vorhandene Konfigurationen zu laden.

4 Konzeptioneller Entwurf

In diesem Kapitel werden Konzepte zur Umsetzung der Betriebs- und Konfigurationssoftware des Datenloggers erarbeitet. Darüber hinaus beinhaltet dieses Kapitel die übergeordneten Konzeptthemen Speichermedium, Schnittstellendefinition und Speicherformat. Die Einführung in den konzeptionellen Entwurf stellen die Randbedingungen zur Umsetzung des Projektes dar.

4.1 Randbedingungen

Zusätzlich zu den unter Kapitel 3 definierten Anforderungen müssen bei der Umsetzung einige weitere Randbedingungen beachtet werden. Wichtige Randbedingungen sind Vorgaben zur Programmiersprache und zur verwendeten Hardware. Eine weitere wichtige Randbedingung für die Implementierung ist die maximal mögliche Verarbeitungszeit je CAN-Botschaft. Diese Randbedingung ist insbesondere für die Umsetzung der Betriebssoftware und bei der Auswahl des Speicherformats relevant.

4.1.1 Maximale Verarbeitungszeit je CAN-Botschaft

Die maximale, für die Verarbeitung einer CAN-Botschaft, zur Verfügung stehende Zeit ist die Zeit zwischen dem Empfangen einer Botschaft bis zum Empfangen der nächsten Botschaft. Die maximale Verarbeitungszeit entspricht demnach der minimalen Übertragungszeit einer CAN-Botschaft. Die minimale Übertragungszeit tritt bei CAN-Botschaften im Standard-Format mit 0 Datenbytes auf. Bei einer solchen Botschaft werden insgesamt 47 Bit übertragen. Bei Übertragung mit der maximalen Busgeschwindigkeit von 1 Mbit/s Sekunde ergibt sich eine Übertragungsdauer von $\frac{47bit}{1Mbits/s} = 47\mu s$.

Bei mehreren an den Datenlogger angeschlossen CAN-Schnittstellen, ergibt sich die maximal mögliche Verarbeitungszeit aus der minimalen Übertragungsdauer dividiert durch die Anzahl der Schnittstellen. Die berechnete Übertragungsdauer stellt die theoretisch kleinste mögliche Zeit zwischen zwei CAN-Botschaften dar. In realen Systemen wird meistens nur eine Bitrate von maximal 500 kbit\s verwendet und zwischen den einzelnen Botschaften existiert im Normalfall eine Pausenzeit, in der keine Übertragung erfolgt [2].

4.1.2 Vorgaben Hardware und Programmiersprache

Als Hardware-Basis für die Betriebssoftware ist die Bertrandt eigene b.embed-Plattform zu nutzen. Dieses Entwicklungsboard stellt einen frühen Prototyp der finalen Hardware dar. Auf der b.embed-Plattform wird der Mikrocontroller EA LPC4088 QuickStart Board verwendet. Dieser Controller verfügt bereits über eine Reihe von On-chip-Peripheriekomponenten wie USB und Ethernet. Zusätzlich sind bereits diverse Speichererweiterung wie SRAM und Flashspeicher auf dem EA LPC4088 QuickStart Board verbaut. Abbildung 4 zeigt die Belegung der Input/Output-Pins des EA LPC4088 QuickStart Boards [8].

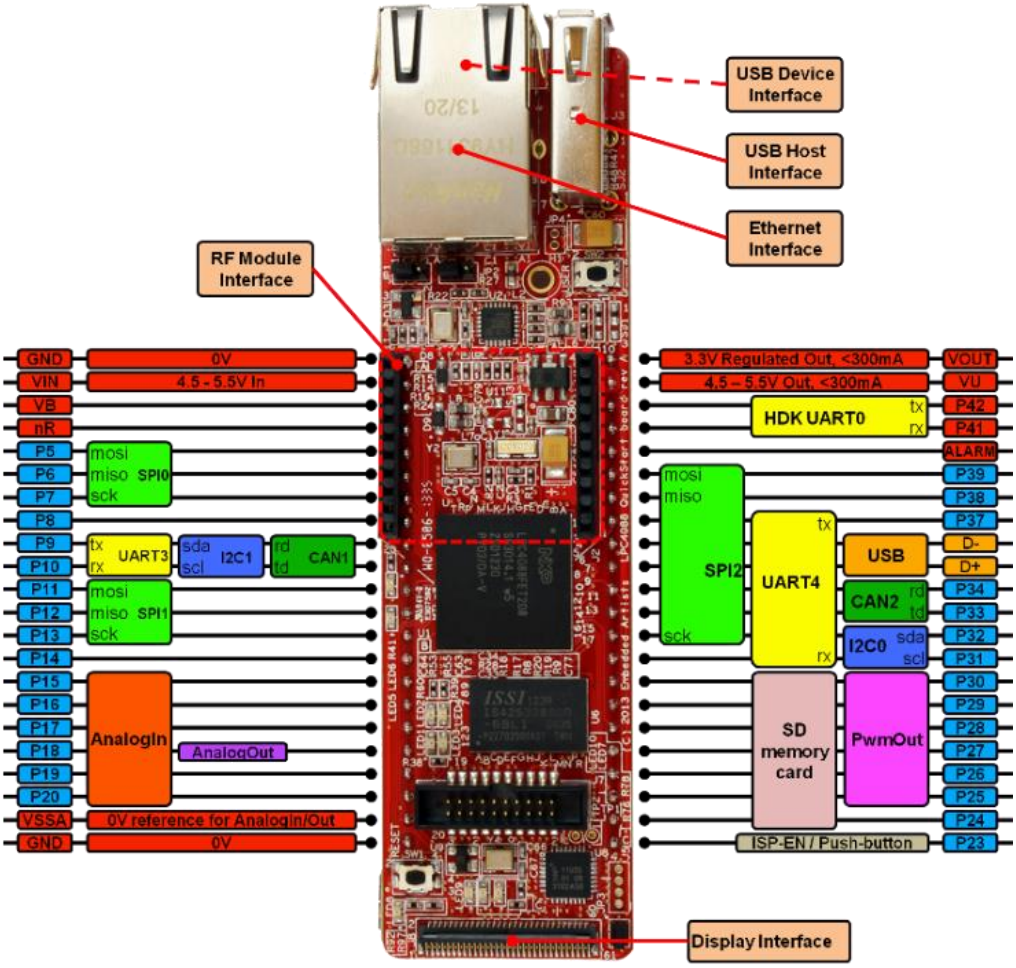

Abbildung 4 - EA LPC4088 QuickStart Board [8]

Bei dem Quickstartboard handelt es sich um einen zum mbed-Framework kompatiblen Mikrocontroller. Das von der Firma ARM Limited entwickelte Framework bietet diverse Softwarebibliotheken zur Nutzung der Peripherekomponeten des Quickstartbords. Bei der Entwicklung der Betriebssoftware soll das mbed-Framework verwendet werden [9]. Die Umsetzung der Software erfolgt in C++ mit Hilfe der vom Hersteller NXP bereitgestellten Entwicklungsumgebung LPCXpresso.

4.2 Übergeordnete Konzeptthemen

In diesem Abschnitt werden die Konzeptthemen Speicherformat, Speichermedium und Schnittstellen behandelt, diese drei Themen betreffen sowohl die Betriebs- als auch die Konfigurationssoftware.

4.2.1 Speicherformat

Für die Auswertung der vom Datenlogger aufgenommenen Daten soll die Software CANoe der Firma Vector Informatik eingesetzt werden, da diese bereits im Unternehmen vorhanden ist und somit zusätzliche Entwicklungskosten für eine eigene Analysesoftware entfallen. CANoe kann die Dateiformate Binary Logging Format und ASCII Logging Files verarbeiten. Diese Dateiformate sind Eigenentwicklungen der Firma Vector Informatik. Zu der Struktur dieser Dateiformate existiert keine öffentlich zugängliche Dokumentation. Folglich lässt sich der Aufbau dieser beiden Dateiformate nur über eine Analyse ermitteln. Bei dem Binary Logging Format handelt es sich um ein Binärformat, welches die Analyse der Syntax erschwert. Die ASCII Logging Files sind ASCII-kodiert und daher menschenlesbar. Diese Tatsache erleichtert die Analyse der Syntax wesentlich. Aus diesen Grund wird die Analyse der ASCII Logging Files gewählt [10].

Abbildung 5 zeigt den Inhalt einer ASC-Datei.

```
1  date Mon Mai 19 15:37:13.000 2014
2  base hex  timestamps absolute
3  internal events logged
4  // version 8.5.0
5  Begin Triggerblock Mon Mai 19 15:37:13.690 2014
6     0.590000 1  539             Rx   d 1 03  Length = 108000 BitCount = 58 ID = 1337
7     0.619000 1  71x             Rx   d 1 42  Length = 152000 BitCount = 80 ID = 113x
8     0.690000 1  539             Rx   d 1 04  Length = 106000 BitCount = 57 ID = 1337
9     0.690000    log trigger event
10    0.719000 1  71x             Rx   d 1 42  Length = 152000 BitCount = 80 ID = 113x
11 End TriggerBlock
12
```

Abbildung 5 - Inhalt einer ASC-Datei

Die ersten vier Zeilen stellen den Kopf der Datei dar und sind, bis auf die Datum- und Uhrzeitangaben, in jeder Datei identisch. Nach dem Start-Tag „Beginn Triggerblock“ folgen die Daten der Aufzeichnung in strukturierter Form. Das End-Tag „End TriggerBlock“ signalisiert das Ende der Aufzeichnung. Die in Abbildung 6 gezeigte Zeile einer ASC-Datei repräsentiert den Datensatz einer CAN-Botschaft.

Abbildung 6 - ASC CAN-Datensatz

Die einzelnen Bestandteile eines Datensatzes und deren Bedeutung werden im Folgenden, anhand der Nummerierung aus Abbildung 6, beschrieben:

1. Vergangene Zeit seit Beginn der Aufzeichnung in Sekunden unter Angabe von 6 Nachkommastellen
2. Die Nummer des CAN-Interfaces, welches die Botschaft empfangen oder versendet hat
3. ID der versendeten/empfangenen Botschaft als hexadezimaler Wert. Bei Extended-CAN wird hinter der ID zusätzlich ein „x“ angehängt
4. Gibt an, ob die Nachricht versendet (Tx) oder empfangen (Rx) wurde
5. Kennzeichnet ob es sich um einen Datenframe(d) oder ein Remoteframe(r) handelt. Bei Datenframes wird zusätzlich die Anzahl der Datenbytes angeben (0-8)
6. Hexadezimale Werte der Datenbytes (Bei mehreren Datenbytes durch Leerzeichen getrennt)
7. Übertragungsdauer in Nanosekunden
8. Anzahl der übertragenen Bits (inklusive notwendiger Stuffingbits)
9. ID der versendeten/empfangenen Botschaft als dezimaler Wert. Bei Extended-CAN wird hinter der ID zusätzlich ein „x“ angehängt

Neben den oben beschrieben CAN-Datensätzen kann eine ASC-Datei Zeilen enthalten, welche das Auslösen eines Triggers kennzeichnen. Eine entsprechende Zeile ist in der Abbildung 7 dargestellt.

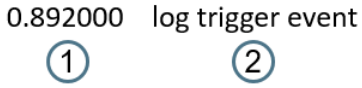

Abbildung 7- ASC Trigger-Kennzeichnung

Eine Zeile zur Kennzeichnung eines ausgelösten Triggers besteht lediglich aus zwei Elementen. Das erste Element enthält die Zeit seit Beginn der Aufzeichnung. Durch das zweite Element erfolgt die eigentliche Kennzeichnung, dass es sich um einen ausgelösten Trigger handelt. Wurde ein Trigger durch eine CAN-Botschaft ausgelöst, ist die Trigger-Kennzeichnung unter dem jeweiligen Datensatz der auslösenden Botschaft zu finden.

Da die Syntax des ASC-Formats vollständig analysiert werden konnte, wäre es möglich, dass der Datenlogger dieses Format zur Aufzeichnung nutzt. Allerdings bringt die Speicherung von ASCII-kodierten Daten Nachteile hinsichtlich Performance und Dateigröße mit sich. Eine Alternative ist die Speicherung in einem binären Speicherformat und die nachträgliche Konvertierung der Daten in der Konfigurationssoftware des Datenloggers.

Um einen Vergleich zwischen dem ASC-Format und einem binären Format erstellen zu können, wird im Folgenden zunächst das binäre Format spezifiziert. Das binäre Format muss, um später in das ASC-Format konvertiert werden zu können, die in Tabelle 2 zusehenden Attribute je Datensatz enthalten. Zusätzlich muss in jeder Datei das Datum und die

Zeit des Aufzeichnungsstarts gespeichert werden. Dieses erfolgt am Anfang der Datei in Form eines Timestamps.

Tabelle 2 - Datensatz im binären Format

8 Byte	1 Byte	2 Byte	4 Byte	1 Byte	1 Byte	1 Byte	8 Byte
Logtime	Interface	Bitrate	ID	Trigger	Typ	Length	Data
Zeit seit Beginn der Aufzeichnung in Mikrosekunden	Nummer des aufzeichnenden Interfaces	Bitrate des Interfaces in kbit/s	ID der CAN-Botschaft	Erfüllt die Botschaft eine Trigger-Bedingung? (0=nein, 1=ja, 2= es handelt sich um einen manuell hinzugefügten Trigger)	0 = Standard-CAN, 1 = Extended-CAN	Anzahl der Datenbytes der Botschaft (0-8)	Datenbyte der Botschaft (unabhängig von Length 8 Byte Reservierung)

Jeder Datensatz einer Botschaft, im in Tabelle 2 gezeigten Format, ist exakt 26 Byte lang. Durch die einheitliche Länge wird bei der späteren Verarbeitung der Daten die Trennung der Datensätze erleichtert. Im binären Format werden einige im ASC-Format vorhandene Attribute wie *Length* und *Bitcount* nicht berücksichtigt, da diese aus anderen Attributen berechnet werden können.

Der Speicherbedarf eines Datensatzes im ASC-Format berechnet sich aus der Anzahl der Zeichen der Zeile multipliziert mit einem Byte, da für die Kodierung jedes ASCII-Zeichen 8 Bit benötigt werden. Der Datensatz in Zeile 6 der Abbildung 5 besteht inklusive Leerzeichen aus 83 Zeichen, also insgesamt 83 Byte. Der Speicherbedarf dieser Zeile im ASC-Format ist damit um Faktor drei größer als im Binär-Format. Da die Länge eines Datensatzes im ASC-Format nicht konstant ist, kann sich dieser Faktor weiter verschlechtern, z.B. bei Botschaften, welche mehr als ein Datenbyte besitzen.

Neben dem benötigten Speicherbedarf pro Datensatz ist ein weiterer wichtiger Faktor die Rechenzeit die zum Schreiben eines Datensatzes benötigt wird. Die Ermittlung der benötigten Rechenzeit erfolgt mittels einer prototypischen Implementierung einer Software, die jeweils einen Datensatz in beiden Formaten auf einem USB-Stick schreibt. Zum Schreiben des Datensatzes wird die Funktion *printf* für das ASC-Format und *fwrite* für das binäre Format verwendet. Die gemessene Verarbeitungszeit beträgt für das ASC-Format im 78 µs und für das binäre Format 32 µs. Das Schreiben eines Datensatzes im binären Format benötigt demnach weniger als die Hälfte der Rechenzeit eines Schreibvorganges für einen Datensatz im ASC-Format. Aufgrund des geringeren Speicherplatzes und der geringeren

benötigten Rechenzeit für das Schreiben eines Datensatzes wird das binäre Format als Speicherformat ausgewählt. Die Auswahl dieses Formats bedingt eine spätere Konvertierung, die dafür benötigte Rechenzeit kann allerdings vernachlässig werden, da die Konvertierung im Gegensatz zur Aufzeichnung kein zeitkritischer Vorgang ist.

4.2.2 Speichermedium

Das Speichermedium soll ohne zusätzlichen Controller an den Mikrocontroller angebunden werden. Das EA LPC4088 QuickStart Board bietet die Möglichkeit zur Anbindung einer SD-Karte, oder eines USB-Speichermediums. In diesem Abschnitt erfolgt die Evaluation der möglichen Speichermedien. Zur Bewertung der beiden Speichermedien werden die folgenden Faktoren herangezogen:

- Die Kosten pro Medium bei einer Speicherkapazität von 16 GB
- Der notwendigere Verschaltungsaufwand auf der Platine
- Verfügbarkeit der Schnittstelle für den Anschluss des Mediums bei PCs und Notebooks
- Verfügbarkeit einer Software-Bibliothek zum Schreiben und Lesen des Mediums

Die Schreib- und Leseraten können bei der Bewertung vernachlässigt werden, da die maximale Datenrate bei CAN-Bus bei 1 Mbit/s beträgt. Selbst SD-Karten und USB-Sticks mit niedrigen Schreibraten von 30 Mbits/s genügen, um die Daten mehrerer CAN-Schnittstellen aufzuzeichnen. Der Kostenvergleich erfolgt anhand von Speichermedien mit einer Kapazität von 16 GB. Diese Speicherkapazität ist bei Verwendung des binären Speicherformats ausreichend um einen Arbeitstag (8 Stunden) aufzuzeichnen.

Tabelle 3 - Bewertung Speichermedien

	SD-Karte	**USB-Stick**
Kosten für 16GB	ca. 15 € [11]	ca. 10 € [12]
Verschaltungsaufwand	Mittel (zusätzliche Widerstände, Kondensatoren und SD-Karten-Aufnahme auf der Platine notwendig)	Keiner (USB-Buchse bereits am Quickstart Board vorhanden)
Verfügbarkeit Schnittstelle	Niedrig (Bei den meisten Notebooks ist ein SD-Karten-Slot vorhanden, aber bei PCs ist meistens ein externer Kartenleser notwendig)	Sehr gut (Standard bei PCs bzw. Notebooks)
Software-Bibliothek vorhanden	Ja [13]	Ja [14]

Tabelle 3 zeigt die vier genannten Bewertungsfaktoren für die Varianten SD-Karte und USB-Stick. Auf Basis dieser Bewertungstabelle fällt die Entscheidung auf einen USB-Stick als Speichermedium. Die ausschlaggebenden Gründe hierfür sind, dass die USB-Schnittstelle zum Standard bei fast allen PCs und Notebooks gehört und somit ein Zusatzgerät zum Anschluss des Speichermediums entfällt. Ein weiterer Grund ist, dass bei Nutzung einer SD-Karte zusätzlicher Verschaltungsaufwand auf der Platine der zukünftigen b.cube Platine erforderlich ist. Dieser kann durch Nutzung eines USB-Sticks entfallen und senkt somit die Komplexität des Platinenentwurfs und die Herstellungskosten. Außerdem blockiert die Anbindung einer SD-Karte sieben Pins des Mircocontrollers (siehe Quickstart Board Pinning Abbildung 4). Diese sieben Pins können bei Verwendung eines USB-Sticks weiterhin für andere Funktionen genutzt werden. Zusammenfassend stellt sich der USB-Stick als bestes Speichermedium für den hier vorliegenden Anwendungsfall heraus.

4.2.3 Schnittstellendefinition

Dieser Abschnitt beschreibt die Schnittstellen zwischen PC und Datenlogger. Die Schnittstelle zum Austausch von aufgezeichneten Daten, mittels einer Datei, ist bereits unter Kapitel 4.2.1 definiert. Im Folgenden soll nun die Schnittstelle zur Konfiguration des Datenloggers definiert werden. Der Datenlogger soll per Android-App und per Windowsapplikation konfigurierbar sein. Für die Android-App wurde bereits in einer früheren Studienarbeit Bluetooth als Kommunikationsschnittstelle zwischen App und Datenlogger festgelegt. Die Kommunikation zwischen der Windowsanwendung und dem Datenlogger könnte ebenfalls per Bluetooth realisiert werden, sodass beide Konfigurationsgeräte über dieselbe Schnittstelle mit dem Datenlogger kommunizieren. Allerdings ist in vielen Unternehmen die Nutzung von drahtlosen PC-Schnittstellen nicht, oder nur mit Sondergenehmigung erlaubt. Aus diesem Grund wird festgelegt, dass die Windowsapplikation mittels einer kabelgebundenen Verbindung mit dem Datenlogger kommunizieren soll. Als Schnittstelle bietet sich die USB-Schnittstelle an, da diese an allen handelsüblichen PCs und Notebooks vorhanden ist. Durch die Verwendung eines USB-Seriell-Konverters kann dasselbe Softwaremodul wie bei Bluetooth verwendet werden, da die Bluetooth Kommunikation ebenfalls über eine serielle Schnittstelle erfolgt. Die Abbildung 8 zeigt die Hardwaresicht des Teilsystems.

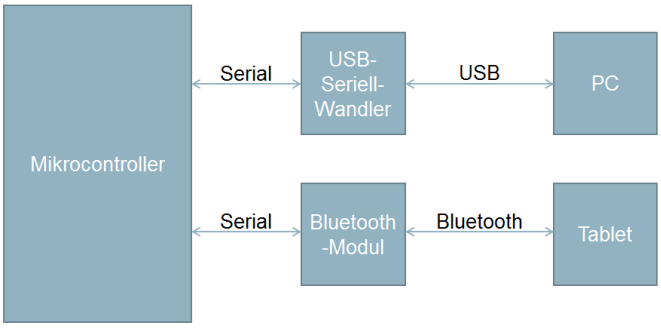

Abbildung 8 - Hardwaresicht Konfigurationsschnittstelle

Die Umschaltung zwischen USB- und Bluetooth-Konfiguration könnte in der späteren Hardwareversion durch einen Hardwareumschalter realisiert werden. Alternativ kann eine solche Umschaltung auch per Software implementiert werden.

Die Konfiguration über die serielle Schnittstelle erfolgt mittels ASCII-kodierten Kommandos. Die einzelnen Kommandos bestehen aus einem oder mehreren Zeichen, welche die Art des Kommandos und den eigentlichen Konfigurationswert beschreiben. Die Konfigurationswerte werden als hexadezimale Zeichenkette übertragen. Für die Kodierung eines Bytes werden jeweils zwei ASCII-Zeichen übertragen. Die Kodierung wird aus der bereits bestehenden Android-App übernommen, um die Komptabilität zwischen den beiden Anwendungen zu gewährleisten. Die Tabelle 4 zeigt die Kommandos, welche von der Konfigurationssoftware an die Betriebssoftware gesendet werden können und die von der Betriebssoftware zu erwartenden Antworten.

Tabelle 4- Kommandotabelle

Zeichen 1	Zeichen 2	Zeichen 3	Bedeutung	Bedeutung / Anzahl Datenbytes	Erwartete Anwort
r			Aufzeichnung starten	-	!
s	-	-	Aufzeichnung stoppen	-	!
i	c	-	Abfrage Anzahl der vorhandenen CAN-Interfaces	-	Anzahl Schnittstellen (1Byte)
k	m	-	Betriebsmodus Konfiguration	Modus (1 Byte / 2 Zeichen) 0 = Permanent 1 = Trigger	! oder r bei aktiver Aufzeichnung
k	a	-	AutoStart/Stopp Konfiguration	1 Byte (2 Zeichen) 0 = aus 1 = ein	! oder r bei aktiver Aufzeichnung
k	c	-	Uhrzeit Konfiguration	Timestamp (4 Byte / 8 Zeichen)	! oder r bei aktiver Aufzeichnung
k	i	-	CAN-Interfaces Konfiguration	Interface-Nummer (1 Byte / 2 Zeichen) aktiviert/deaktiviert (1 Byte / 2 Zeichen) 0 = deaktiviert 1 = aktiviert Bitrate (2 Byte / 4 Zeichen)	! oder r bei aktiver Aufzeichnung
k	t	r	Trigger zurücksetzten	-	! oder r bei aktiver Aufzeichnung
k	t	a	Trigger hinzufügen	ID (4 Byte / 8 Zeichen) StartByte (1 Byte / 2 Zeichen) StartBit (1 Byte / 2 Zeichen) Länge (1 Byte / 2 Zeichen) Operation (1 Byte / 2 Zeichen) Referenzuwert (8 Byte / 16 Zeichen)	! oder r bei aktiver Aufzeichnung
k	e	-	Ende der Konfiguration	-	! oder r bei aktiver Aufzeichnung

Auf ein empfangenes Kommando antwortet der Mikrocontroller mit einer definierten Antwort. Bei allen mit *k* beginnenden Kommandos kann die Betriebssoftware alternativ mit einem *r* antworten. Empfängt die Konfigurationssoftware ein *r* als Antwort ist, eine Aufzeichnung aktiv und es kann keine Übernahme der Konfiguration durch die Betriebssoftware erfolgen. Sollte die Antwort des Mikrocontrollers nicht innerhalb einer bestimmten Zeit vorliegen, oder sendet der Mikrocontroller eine nicht erwartete Antwort, kann die Konfigurationssoftware eine Störung der Kommunikation erkennen und den Anwender informieren.

Abbildung 9 zeigt den prinzipiellen Kommunikationsablauf zwischen Betriebs und Konfigurationssoftware während der Konfiguration des Datenloggers.

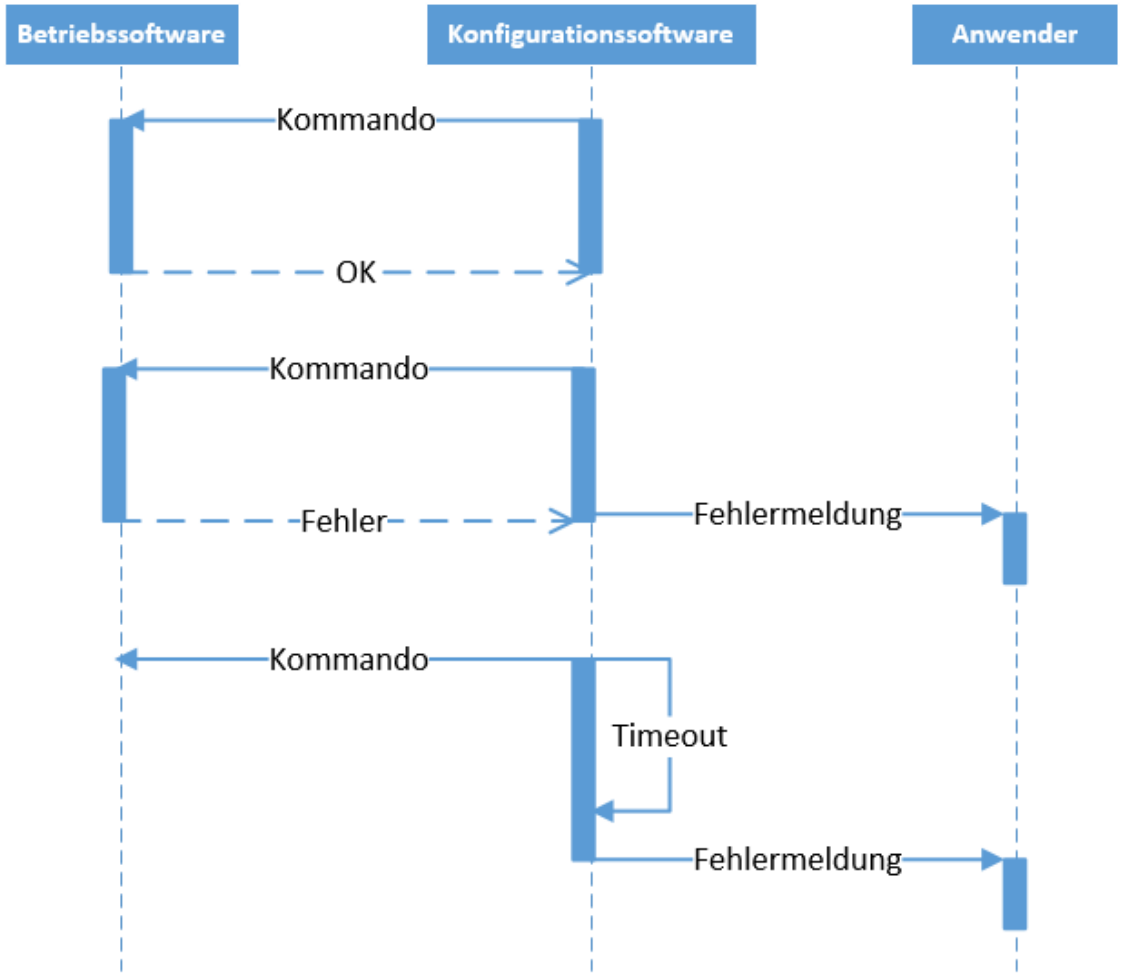

Abbildung 9 - Kommunikationsablauf

Das erste von der Konfigurationssoftware an die Betriebssoftware gesendete Kommando wird korrekt bestätigt. Bei dem zweiten übertragenen Kommando erhält die Konfigurationssoftware eine falsche Antwort vom Mikrocontroller und gibt eine Fehlermeldung an den Anwender aus. Auf das dritte Kommando erhält die Konfigurationssoftware nicht rechtzeitig eine Antwort und gibt deswegen ebenfalls eine Fehlermeldung an den Anwender aus.

4.3 Betriebssoftware-Konzept

Aus den unter Kapitel 3 definierten Anforderungen ergeben sich die folgenden, durch die Betriebssoftware zu realisierenden Hauptfunktionen:

- USB-Stick-Anbindung
- Anbindung, Konfiguration und Auslesen von CAN-Schnittstellen
- Empfang und Umsetzung von Kommandos der Konfigurationssoftware
- Überprüfung von CAN-Botschaften auf Trigger-Bedingungen
- Realisierung eines Pufferspeichers für den Vorlauf im Trigger-Modus
- Starten und Stoppen der Aufzeichnung
- Verarbeitung und Speicherung von CAN-Botschaften
- Speichern der Konfiguration in einem nichtflüchtigen Speicher

In den folgenden Abschnitten werden diese Teilfunktionen genauer betrachtet und entsprechende Lösungsmöglichkeiten erarbeitet. Am Ende dieses Kapitels wird, auf Basis der Lösungsansätze der einzelnen Teilfunktionen, ein erster Entwurf der Struktur der Betriebssoftware erstellt.

4.3.1 USB-Stick-Anbindung

Für das EA LPC4088 QuickStart Board existiert eine Bibliothek zum Speichern und Lesen von Daten auf einen USB-Stick. Die LPC4088-USBHost Bibliothek ermöglicht den Zugriff auf einen FAT16, oder FAT32-formatierten Datenträger. Das FAT32-Dateisystem ermöglicht eine maximale Dateigröße von 4 GB, weswegen möglicherweise bei längeren Logging-Vorgängen eine Aufteilung der Aufzeichnung auf mehrere Dateien notwendig ist. Außerdem muss eine Prüfung implementiert werden, ob die Speicherkapazität des USB-Sticks erschöpft ist. Sollte auf Grund von mangelnder Speicherkapazität keine weitere Aufzeichnung möglich sein, muss der Anwender informiert werden. Die Benutzung der LPC4088-USB-Host Bibliothek erleichtert die Implementierung der USB-Stick-Anbindung erheblich, da ansonsten eine eigene Implementierung des Dateisystems notwendig wäre [14].

4.3.2 Anbindung, Konfiguration und Auslesen von CAN-Schnittstellen

Für die Anbindung von CAN-Schnittstellen stellt das mbed-Framework die Klasse *CAN* zur Verfügung. Die Klasse verfügt bereits über die geforderte Methode zur Einstellung der Bitrate des Interfaces (Anforderung A02). Die Klasse *CAN* ermöglicht es, mittels der Methode *read,* CAN-Botschaften in ein Objekt von Typ *CANMessage* einzulesen. Das Auslesen kann entweder mittels Polling, oder durch Implementierung einer Interrupt Service Routine (ISR) für Reaktion auf empfangene Botschaften erfolgen [15]. Das Einlesen mittels Polling hat den Nachteil, dass das CAN-Interface ständig abgefragt werden muss, ob eine neue CAN-Botschaft vorliegt. Außerdem verfügt die CAN-Schnittstelle über einen begrenzten Emp-

fangspuffer. Werden die Botschaft zu langsam ausgelesen werden die Botschaften im Empfangspuffer des Interfaces überschrieben. Dieses führt zum Verlust von Botschaften. Daher ist es sinnvoll, das Auslesen der Botschaften innerhalb einer ISR zu implementieren, welche bei Empfang einer Botschaft aufgerufen wird. Durch die Nutzung einer ISR wird das Risiko eines Botschaftsverlustes minimiert. Damit das System nicht dauerhaft durch das Abarbeiten von ISRs blockiert wird, muss die Zeit zur Abarbeitung der ISR möglichst klein gehalten werden. Daher wird vorgesehen, in der ISR die empfangenen Botschaften lediglich auszulesen und zur späteren Verarbeitung zwischenzuspeichern. Die Verarbeitung der eingelesenen Botschaften sowie deren Speicherung auf dem USB-Stick kann außerhalb der ISR erfolgen.

Die Anzahl der Speicherplätze des hierfür zu implementierenden Zwischenspeichers kann klein dimensioniert werden, da die Verarbeitungszeit für eine zwischengespeicherte Botschaft kleiner der Zeit zwischen dem Einlesen zweier Botschaften sein muss. Sollte die Verarbeitungszeit ständig größer als die Zeit zwischen den einzelnen empfangenen Botschaften sein, würde es, unabhängig von der Größe des Puffers zu dessen Überlauf kommen. Eine Kapazität von einigen hundert Elementen sollte ausreichen, um bei temporär auftretender Prozessorbelastung die Vollständigkeit der Daten und deren Konsistenz sicherzustellen. Für die Umsetzung des Zwischenspeichers wird ein 2^n-Ringbuffer gewählt. Der 2^n-Ringbuffer funktioniert nach dem First-in-First-Out(FIFO)-Prinzip und wird auf Grund seiner guten Performance gewählt [16]. Ein weiterer Vorteil der Implementierung eines Interrupts für die einzelnen CAN-Schnittstellen ist, dass durch Deaktivierung des Interrupts die Aufzeichnung einer CAN-Schnittstelle deaktiviert werden kann, da nach der Deaktivierung keine Botschaften von dem entsprechenden Interface eingelesen werden.

Wie zu Beginn dieses Abschnitts erwähnt, werden die CAN-Botschaften in ein Objekt vom Typ *CANMessag*e eingelesen. Ein Objekt dieses Typs enthält die folgenden, in Tabelle 5 dargestellten, Daten einer CAN-Botschaft.

Tabelle 5 - Attribute eines CANMessage-Objektes

Attribut	Type	Beschreibung
id	unsigned int	Identifier der Botschaft
data[8]	unsigned char	Nutzdaten der Botschaft
len	unsigned char	Länge (Anzahl) der Nutzdatenbytes
CANFormat	Enum format	0 - Standard CAN-Frame 1 - Extended CAN-Frame
CANType	Enum type	0 - Datenframe 1 - Remote Frame

Für die Aufzeichnung einer CAN-Botschaft, im unter Kapitel 4.2.1 definierten, binären Format fehlen die Angaben zur vergangenen Zeit seit Beginn der Aufzeichnung (Logtime), die Nummer des CAN-Interfaces, welches die Botschaft empfangen hat (Interface) und die Bitrate des Interfaces. Für diese drei Attribute müssen alternative Datenquellen gefunden werden. Für die beiden Attribute Bitrate und Interface ist eine Erweiterung der Klasse *CAN* um diese beiden Attribute sinnvoll. Dieses kann durch Ableitung einer neuen Klasse *CANInterface* von der Klasse *CAN* realisiert werden. Das Attribute *Logtime* kann durch einen Timer realisiert werden, der zu Beginn der Aufzeichnung gestartet wird. Auf diesen Timer können die einzelnen CAN-Interfaces die vergangene Zeit, seit Start der Aufzeichnung, auslesen. Für die Realisierung eines Timers bietet das mbed-Framework die Klasse *Timer* an. Diese Klasse hat allerdings den Nachteil, dass für die Speicherung des Mirkosekunden-Wertes nur eine 32-Bit Variable genutzt wird, dadurch kommt es nach circa einer Stunde und zehn Minuten zum Überlauf des Timers [17]. Um dieses Problem zu beheben, muss die *Timer*-Klasse angepasst und die Variable zum Speichern des Mirkosekunden-Wertes durch eine 64-Bit Variable ersetzt werden. Der in der 64 Bit-Variable speicherbare Wert ist so groß (5849424 Jahre), dass ein Überlauf des Timers ausgeschlosssen werden kann. Eine direkte Anpassung der Klasse *Timer* hätte den Nachteil, dass sobald die eingebundene Bibliothek im Zuge eines Updates aktualisiert wird, eine erneute Anpassung der Klasse erforderlich ist. Sinnvoller ist es eine eigene Klasse zu implementieren, welche eine 64Bit-Vairable zum Speichern des Mirkosekunden-Wertes und ein Objekt der Klasse *Timer* enthält. Zusätzlich muss in der Klasse eine Methode implementiert werden die vor Überlauf des Timers dessen aktuellen Wert ausliest und auf die 64bit-Vairable addiert. Der Überlauf des Timers kann durch einen zyklischen Aufruf dieser Methode nach jeder vergangenen Stunde verhindert werden. Zum zyklischen Aufruf der Methode bietet sich die Klasse *Ticker* des mbed-Frameworks an, welche den zyklischen Aufruf von Methoden nach einer definierten Zeit erlaubt [18]. Zusätzlich muss in der Klasse eine Methode zum Abruf der aktuellen Logtime implementiert werden, da diese nicht direkt aus dem Timer oder der 64 Bit-Variable ausgelesen werden kann, sondern sich aus dem Wert der 64 Bit-Varibale zuzüglich des aktuellen Wertes des Timers, ergibt.

4.3.3 Empfang und Umsetzung von Kommandos der Konfigurationssoftware

Die Betriebssoftware soll von der Konfigurationssoftware Kommandos zur Konfiguration sowie zum Starten und Stoppen der Aufzeichnung über eine serielle Schnittstelle erhalten. Zur Implementierung einer seriellen Schnittstelle bietet das mbed-Framework die Klasse *Serial* an. Einem Objekt der Klasse *Serial* kann eine ISR hinzugefügt werden, welche beim Empfang von Daten ausgerufen wird. Innerhalb der ISR können die Daten, in diesem Fall Kommandos, ausgelesen werden [19]. Die im Abschnitt 4.2.3 definierten Kommandos haben keine einheitliche Länge, darum muss ein zeichenweises Auslesen erfolgen, um festzustellen um welches Kommando es sich handelt. Nach dem Ermitteln des Kommandos kann die entsprechende, zum Kommando gehörende, Anzahl von Datenbytes eingelesen werden.

Die Abbildung 10 zeigt das Prinzip des schrittweisen Einlesens in einem stark vereinfachten Aktivitätsdiagramm, welches aus Gründen der Übersichtlichkeit lediglich einen Teil der Kommandos enthält.

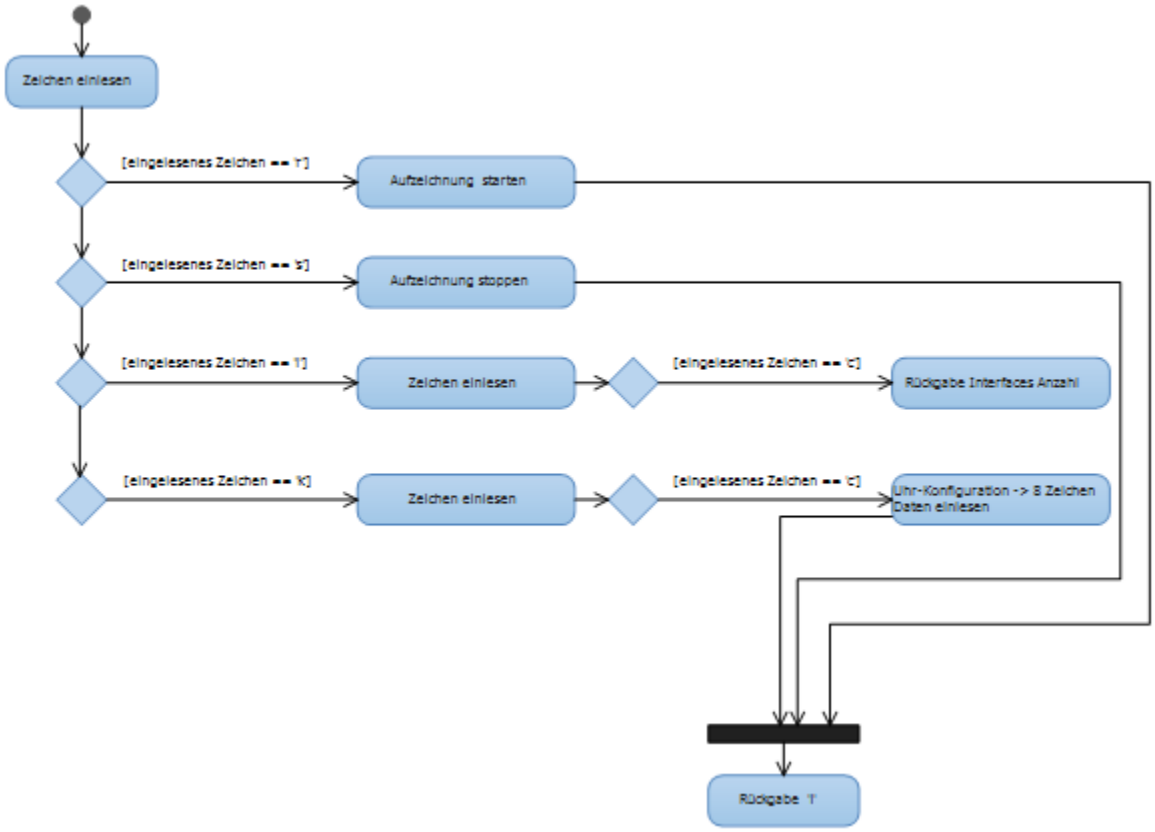

Abbildung 10 - Verarbeitung von empfangenen Konfigurationskommandos

Wie bereits in Abschnitt 4.2.3 beschrieben, werden die Konfigurationsdaten als hexadezimale Zeichenketten übertragen, wobei jeweils zwei Zeichen den Wert eines Bytes repräsentieren. Damit die Betriebssoftware die Daten verarbeiten kann, müssen die hexadezimalen Zeichen in Zahlenwerte konvertiert werden.

4.3.4 Überprüfung von CAN-Botschaften auf Trigger-Bedingungen

Über die Konfigurationssoftware hat der Anwender die Möglichkeit Trigger-Bedingungen zu definieren, welche sich auf ein Signal einer CAN-Botschaft (Kapitel 2) beziehen. Trifft eine solche Bedingung auf eine empfangene Botschaft zu, sind alle Botschaften die zwei Sekunden vor und nach dieser Botschaft empfangen werden, aufzuzeichnen. Die vom Anwender über die Konfigurationssoftware konfigurierten Trigger-Bedingungen müssen nach der Übertragung an den Datenlogger durch die Betriebssoftware zwischengespeichert werden.

Es ist sinnvoll, die Anzahl der durch den Anwender konfigurierbaren Trigger zu beschränken und somit auch die Größe des Zwischenspeichers. Für die Implementierung wird eine

maximale Anzahl von 50 Triggern vorgesehen. Bei einer unbegrenzten Anzahl von Triggern ist es nicht möglich, die Zeit für die Überprüfung einer Botschaft auf Trigger-Bedingungen zu garantieren. Dies könnte zu Folge haben, dass die Verarbeitungszeit einer Botschaft zu groß wird. Wird die Verarbeitungszeit größer den Zeitraum zwischen zwei empfangenen Botschaften, kommt es zu einem Überlauf des Zwischenspeichers für empfangene Botschaften und somit zum Datenverlust. Ob die Verarbeitungszeit bei maximal 50 Trigger-Bedingungen innerhalb der Toleranz liegt, muss in der Testphase geprüft werden. Sollte die Verarbeitungszeit zu groß sein, muss die Anzahl der Trigger-Bedingungen weiter reduziert werden.

Um die einzelnen Trigger zu speichern, wird eine Implementierung einer Klasse *Trigger* vorgesehen. Ein Objekt dieser Klasse repräsentiert eine durch den Anwender definierte Trigger-Bedingung. Die Abbildung 11 zeigt das Klassendiagramm der zu implementierenden Klasse.

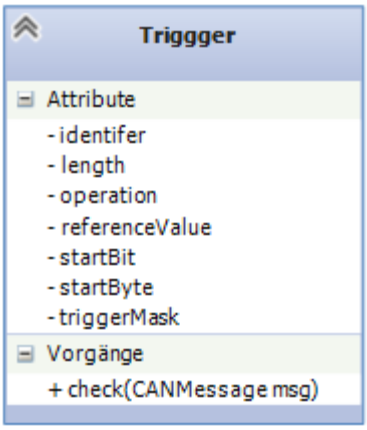

Abbildung 11 - Konzept der Trigger-Klasse der Betriebssoftware

Neben den, die Trigger-Bedingungen beschreibenden Attributen (*identifier*, *length*, *operation*, *referencencValue*, *startBit* und *startByte),* enthält die Klasse zusätzlich das Attribut *trigggerMask.*

Die Variable *triggerMask* enthält die Bitmaske zur Ausmaskierung der nicht zum zu untersuchenden Signal gehörenden Datenbits einer CAN-Botschaft. Für die Erstellung der Triggermaske müssen die folgenden Operation durchgeführt werden:

- Alle Bits der Triggermaske auf 1 setzen
- Bits der Triggermaske um die Länge (*length)* des Signals nach links verschieben
- Bits der Triggermaske invertieren
- Bits der Triggermaske um (*startBit* + *startByte* * 8) Stellen nach links schieben

Bei der Überprüfung, ob es sich bei einer Botschaft um einen Trigger handelt, muss geprüft werden, ob der Wert bestimmter Bits innerhalb der Datenbytes einer CAN-Botschaft eine bestimmte Bedingung in Abhängigkeit zu einem Referenzenwert erfüllt. Beispielweise ob

der Wert der Bits gleich dem Referenzwert ist. Für die Durchführung des Vergleichs werden sowohl die Datenbytes, als auch der Referenzwert als 64 Bit-Variable betrachtet, deren Werte mit einander verglichen werden. Damit der Vergleich ein korrektes Ergebnis liefert, müssen die Bits des Referenzwerte auf die entsprechende Position der zu untersuchenden Bits der Datenbytes verschoben werden. Außerdem müssen die nicht zu untersuchenden Bits der Datenbytes der CAN-Botschaft durch eine UND-Verknüpfung mit der Bitmaske ausmaskiert werden.

Das Prinzip wird im Nachfolgenden anhand eines Beispiels erläutert. In diesem Beispiel soll geprüft werden, ob der Wert eines Signals mit StartByte = 1, StartBit =3 und einer Länge von 2 gleich dem Referenzwert von 3 ist. Zur besseren Übersicht werden nur die Bytes 0-1 dargestellt.

Die Datenbytes der zu untersuchenden CAN-Botschaft haben das folgende Bitmuster:

Tabelle 6 – Datenbytes der zu überprüfenden CAN-Botschaft

Byte 1								Byte 0							
0	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1

Zunächst wird der Referenzwert auf die Position des CAN-Signals verschoben.

Tabelle 7 – Verschiebung des Referenzwert für Triggerprüfung

Byte 1								Byte 0								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	nicht verschobener Referenzwert
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	um (startBit + startByte * 8) verschobener Referenzwert

Der Wert des verschobenen Referenzwertes aus Tabelle 7 beträgt ist 0x1800.

Als nächstes wird die Bitmaske zur Ausmaskierung der nicht zum CAN-Signal gehörenden Datenbits erstellt. Tabelle 8 zeigt das schrittweise Erstellen der Bitmaske.

Tabelle 8 - Erstellung der Trigger-Bitmaske

Byte 1								Byte 0								
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	alle Bits 1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	um die Länge des Signals nach links geschoben
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	invertiert
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	um (startBit + startByte * 8) verschoben

Anschließend erfolgt das Maskieren der Datenbits der CAN-Botschaft durch eine UND-Verknüpfung mit der Bitmaske.

Tabelle 9 - Triggerprüfung maskieren der Datenbits

Byte 1								Byte 0								
0	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1	Datenbytes
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	Bitmaske
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	Ergebnis der UND-Verknüpfung

Die Tabelle 9 zeigt das Ergebnis der UND-Verknüpfung der Datenbytes mit der Bitmaske. Das Ergebnis ist beträgt 0x1800 und ist identisch zu dem Wert des verschobenen Referenzwertes. Somit ist der Wert des Signals gleich 3 und die Trigger-Bedingung ist erfüllt.

Die Abbildung 12 zeigt den für die Tiggerprüfung zu realisierenden Ablauf.

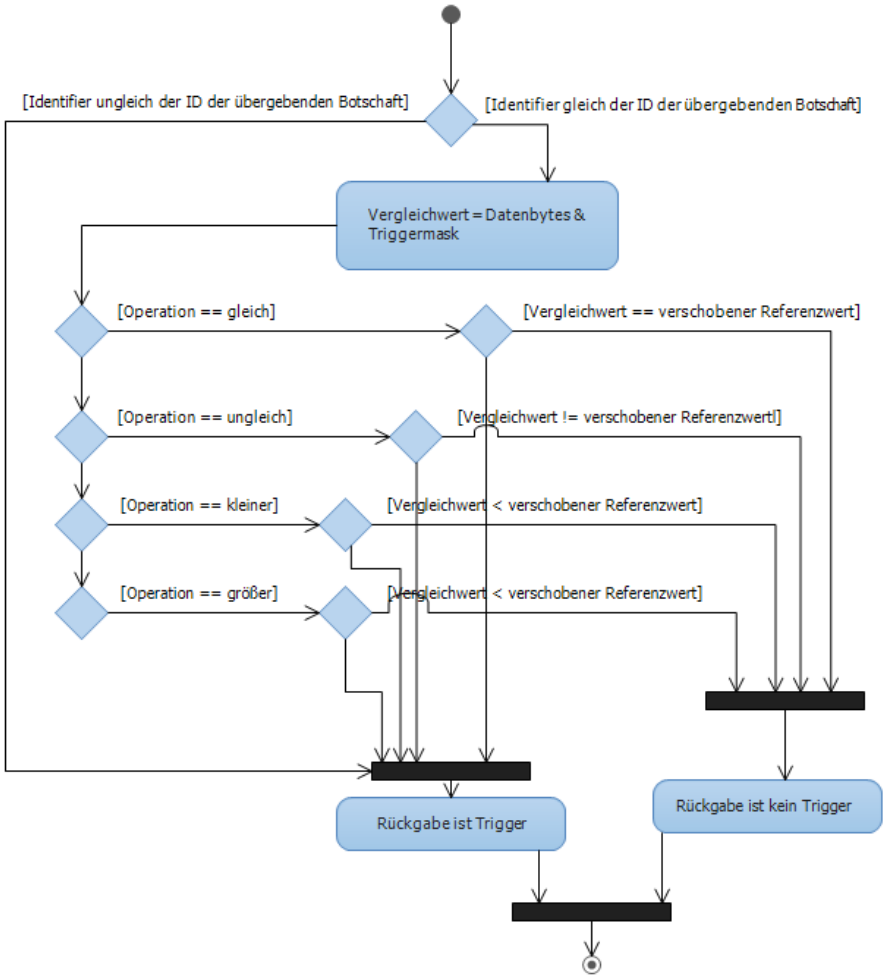

Abbildung 12 - Ablauf Triggerprüfung

Der in Abbildung 12 dargestellten Methode wird eine CAN-Botschaft in Form eines Objektes vom Typ *CANMessage* übergeben. Als Erstes erfolgt eine Überprüfung, ob die ID der CAN-Botschaft mit dem für diesen Trigger hinterlegten Identifier übereinstimmt. Ist das nicht der Fall, ist der Trigger-Bedingung für die übergebende Botschaft nicht gültig und die Überprüfung kann abgebrochen werden. Stimmen die beiden IDs überein, werden die Datenbytes,

der übergebenen Botschaft, mit der Triggermaske verknüpft und in der Variable Vergleichswert zwischengespeichert. Anschließend wird mit Hilfe der *operation*-Variable des Triggers festgelegt welche Bedingung, der Vergleichswert erfüllen muss. Erfüllt der Vergleichwert die Bedingung handelt es sich bei der übergebenen CAN-Botschaft um einen Trigger.

4.3.5 Pufferspeicher für Vorlauf

Befindet sich der Datenlogger im Trigger-Modus, müssen Botschaften für den Vorlauf eines Trigger-Events zwischengespeichert werden. Damit das geeignete Speicherverfahren ausgewählt werden kann, wird zuerst die Anzahl der zu speichernden Datensätze und deren Gesamtspeicherbedarf ermittelt.

Für die Berechnung der Anzahl der Datensätze wird von einer Bitrate des CAN-Interfaces von 500 kbit/s und einer minimalen Länge der Botschaften ausgegangen. Die minimale Länge wird bei Standard-CAN-Frames mit 0 Byte Nutzdaten erreicht, diese Frames sind insgesamt 47 Bit groß. Die 47 Bit ergeben sich wie folgt:

- 1 Bit Start of Frame (SOF)
- 11 Bit Identifier
- 1Bit Remote Transmission Request (RTR)
- 1 Bit Identifier Extension Bit (DIE)
- 1 Bit Reserviert
- 4 Bit Data Lengh Code (DLC)
- 0 Bit Data
- 16 Bit Cyclic Redundancy Check (CRC)
- 2 Bit Acknowledge (ACK)
- 7 Bit End of Frame (EOF)
- 3 Bit Interframe-Space

Daraus ergibt sich eine Übertragungszeit für ein CAN-Frame von $\frac{47\ bit}{500 kbit/s} = 94\ \mu s$. Bei einer minimalen Pausenzeit (Interframe-Space 3 Bit) werden 10638 Frames/Sekunde übertragen [20]. Ausgehend von Basisausstattung des Datenloggers mit zwei CAN-Interfaces und einer Vorlaufzeit von zwei Sekunden ergibt sich eine Anzahl von $2 * 2 * 10638 = 42553$ für den Vorlauf zu speichernde Datensätzen. Für die Speicherung eines Datensatzes im unter Kapitel 4.2.1 beschrieben binären Format werden 26 Bytes benötigt, sodass insgesamt ein Zwischenspeicher mit einer Kapazität von ca. 1,2 Mbyte benötigt wird.

Anhand der benötigten Kapazität kann der Vorlauf sowohl auf den USB-Stick, als auch im 32MB-SRAM des Mikrocontrollers gespeichert werden. Die Speicherung des Vorlaufs auf dem USB-Stick muss in einer separaten Datei erfolgen und ist im Hinblick auf die Performance nicht optimal, da die Schreib- und Lesezugriffe auf den USB-Stick eine längere Zeit in Anspruch nehmen, als Zugriffe auf den SRAM des Mikrocontrollers. Außerdem muss die

temporäre Vorlaufdatei ständig aktualisiert werden. Das Aktualisieren einer Datei benötig deutlich mehr Zeit als z.B. das Aktualisieren eines Arrays im RAM.
Aus diesen Gründen sowie auf Grund der Tatsache, dass es sich bei dem Vorlauf-Puffer um eine interne Komponente handelt, welche für den Anwender unsichtbar sein sollte, stellt der SRAM die optimale Lösung für den Vorlaufpuffer dar.

Für die Umsetzung des Vorlaufs im SRAM biete sich eine Realisierung als Ringspeicher an. Der Ringspeicher verfügt, ausgehend von der obigen Berechnung, über mindestens 42553 Speicherplätze, um einen Vorlauf von zwei Sekunden zur ermöglichen. Sind alle Plätze des Puffers belegt und es soll ein weiterer Datensatz im Vorlauf gespeichert werden, wird der älteste Datensatz durch den Neuen ersetzt. Durch dieses Verfahren befinden sich immer die aktuellsten Datensätze im Vorlauf. Allerdings gilt es zu beachten, dass nicht immer genau zwei Sekunden im Vorlauf sind, da je nach zeitlichem Abstand, zwischen den empfangenen CAN-Botschaften und der Bitrate der CAN-Interfaces eine andere Datensatzanzahl in zwei Sekunden empfangen wird. Von daher ist vor dem Schreiben des Vorlaufs in die Logdatei eine Überprüfung der Empfangszeit eines jeden Elements im Vorlauf-Puffer notwendig.

Für die Realisierung eines Ringspeichers gibt es verschiedene Möglichkeiten, z.B. Arrays und Listen. Für den Vorlauf-Puffer wird eine Implementierung mittels der C++ Klasse *deque* (double-ended queue) vorgesehen. Deque ist ein Container ähnlich, welcher einer Liste ähnelt. Er ermöglicht das schnelle Einfügen und Löschen von Elementen am Anfang und am Ende der „Liste“ [21]. Die Abbildung 13 zeigt die Vorlauf-deque mit markiertem Anfang und Ende.

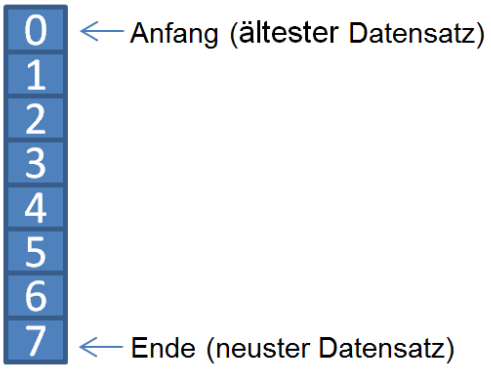

Abbildung 13 - Vorlaufpuffer

Zusätzlich zur der *deque* wird zur Realisierung des Ringpuffers noch eine Intenger-Variable zur Zählung der in der *deque* abgelegten Datensätze benötigt. Ist die Zählvariable gleich der maximalen Anzahl von Datensätzen (42553) und es soll ein neues Element eingefügt werden, wird das Element am Anfang der *deque* gelöscht und das neue Element am Ende eingefügt. Der Ablauf für das Einfügen von Datensätzen ist in dem in Abbildung 14 zu sehenden Aktivitätsdiagramm dargestellt.

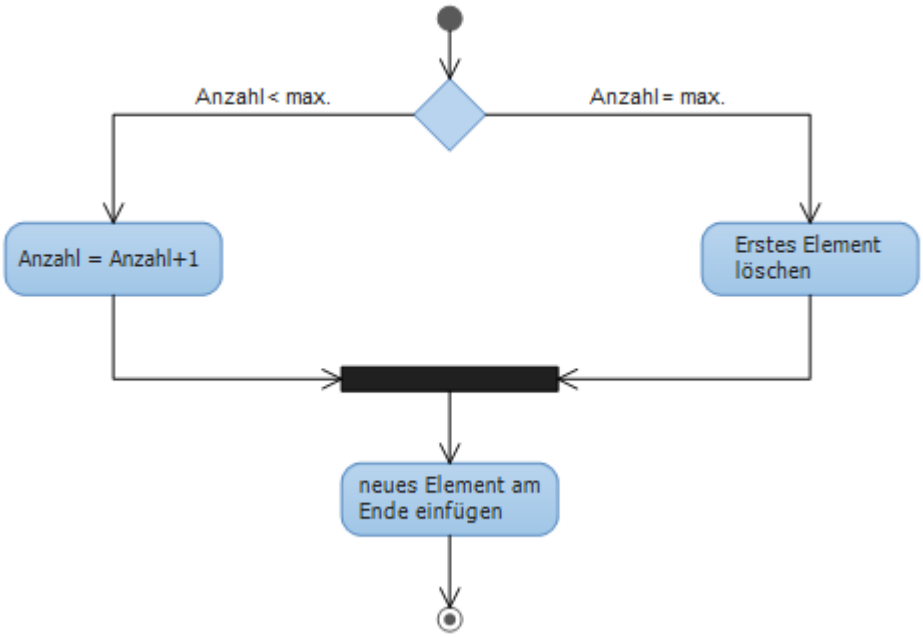

Abbildung 14 - Einfügen von Datensätzen in den Vorlaufpuffer

Bevor das erste (älteste) Element des Vorlaufpuffers durch ein Neues ersetzt wird, muss geprüft werden, ob das erste Element zum Vorlauf einer auf dem USB-Stick zu speichernden Trigger-Bedingung gehört. Ist dies der Fall muss wir das Element zunächst auf dem USB-Stick gespeichert. Im Anschluss wird dieses Element dann überschrieben.

4.3.6 Starten und Stoppen der Aufzeichnung

Die Aufzeichnung von CAN-Botschaften soll entweder manuell, über die Konfigurationssoftware, oder automatisch gestartet und gestoppt werden.

Bevor die Aufzeichnung manuell oder automatisch gestoppt werden kann, muss sichergestellt werden, dass alle Botschaften aus dem Zwischenspeicher für eingehende CAN-Botschaften verarbeitet wurden. Befindet sich der Datenlogger im Trigger-Modus, muss außerdem sichergestellt werden, dass alle relevanten Elemente aus dem Vorlaufspeicher auf den USB-Stick geschrieben werden. Ein sofortiges Abschließen einer Logdatei kann zu Dateninkonsistenz führen.

Darum wird für die Implementierung ein Beenden der Aufzeichnung in zwei Schritten vorgesehen. Das Aktivitätsdiagramm in Abbildung 15 zeigt den groben Ablauf des zweistufigen Stoppens der Aufzeichnung.

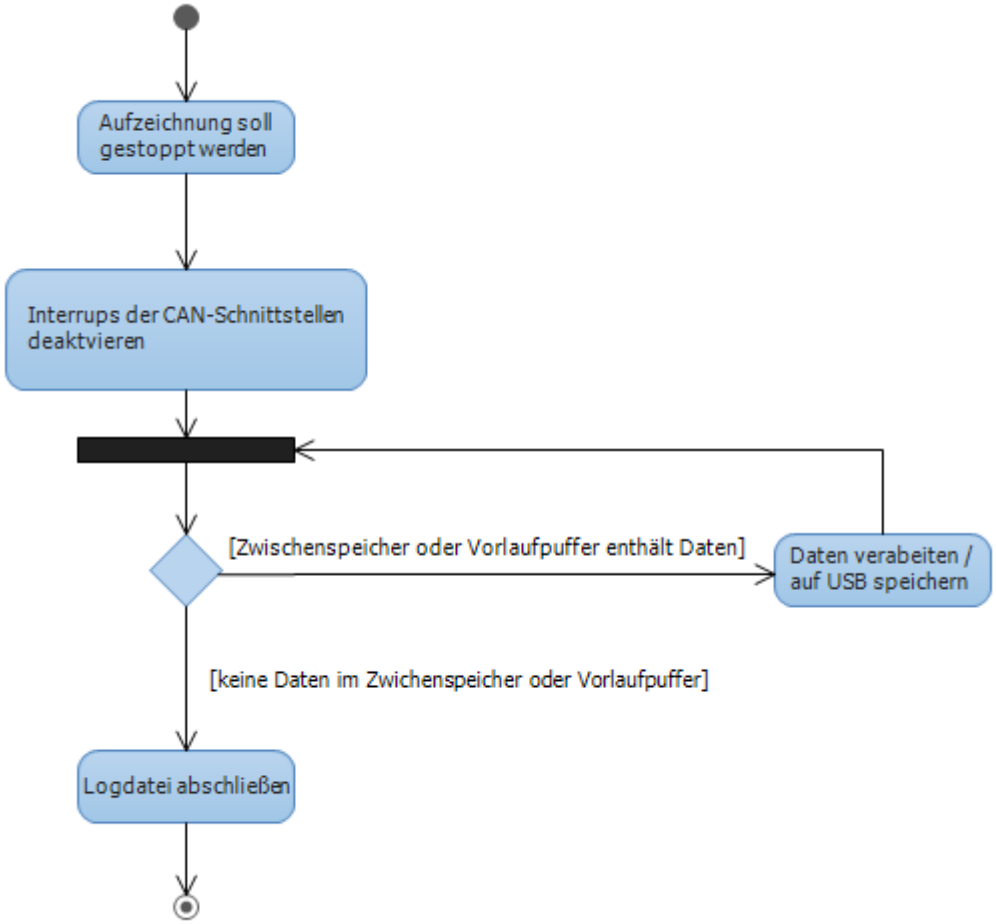

Abbildung 15- Aktivitätsdiagramm „Aufzeichnung stoppen“

Sobald die Aufzeichnung gestoppt werden soll, werden zuerst die Interrupts aller CAN-Schnittstellen deaktiviert, sodass keine neuen Daten in den Zwischenspeicher eingefügt werden. Anschließend erfolgt eine Prüfung, ob im Zwischen- oder Vorlaufspeicher Daten enthalten sind. Ist dieses der Fall, wird der erste Datensatz verarbeitet. Der Ablauf wird solange wiederholt, bis sowohl der Zwischen- als auch der Vorlaufspeicher leer sind. Sind keine Daten mehr vorhanden kann, die Logdatei geschlossen werden.

Um die Aufzeichnung manuell zu starten, ist lediglich ein Aktivieren der Interrupts, der durch den Anwender aktivierten, CAN-Schnittstellen notwendig. Damit zwischen aktiven und deaktivierten CAN-Interfaces unterschieben werden kann, muss ein zusätzliches Attribut in der unter 4.4.3 definierten Klasse *CANInterfaces* vorgesehen werden, indem gespeichert wird ob ein Interface aktiviert oder deaktiviert ist.

Das automatische Starten der Aufzeichnung kann realisiert werden, in dem in der ISR der CAN-Interfaces eine Abfrage implementiert wird in welcher geprüft wird, ob sich der Datenlogger im Automatikmodus befindet und ob eine Aufzeichnung aktiv ist. Empfängt ein Interface bei aktiviertem Automatikmodus eine Botschaft und es ist noch keine Aufzeichnung

aktiv, wird die Aufzeichnung gestartet. Voraussetzung hierfür ist, dass sofern sich der Datenlogger im Automatikmodus befindet, die Interrupts der, durch den Anwender aktivierten, CAN-Schnittstellen aktiviert sind.

Das automatische Stoppen kann durch die Verwendung eines Timers realisiert werden. Der Timer wird beim Empfang einer Botschaft gestartet und bei jedem erneuten Empfang zurückgesetzt. Erreicht der Timer den Wert von einer Minute, wird die Aufzeichnung nach dem weiter oben beschriebenen Verfahren gestoppt. Wichtig ist hierbei, dass nach dem Abschluss der Datei die Interrupts der aktivierten Interfaces wieder aktiviert werden, um ein erneutes automatisches Starten zu ermöglichen.

4.3.7 Verarbeitung und Speicherung von CAN-Botschaften

Dieser Abschnitt behandelt die Verarbeitung und Speicherung der, durch die CAN-Schnittstellen empfangenen und im Zwischenspeicher (*LogBuffer*) abgelegten, CAN-Botschaften. Welche Schritte für die Verarbeitung und Speicherung der CAN-Botschaften durchgeführt werden müssen, ist abhängig von dem Betriebsmodus, in dem sich der Datenlogger befindet.

Befindet sich der Datenlogger im permanenten Aufzeichnungsmodus müssen lediglich die in Abbildung 16 zu sehenden Schritte realisiert werden.

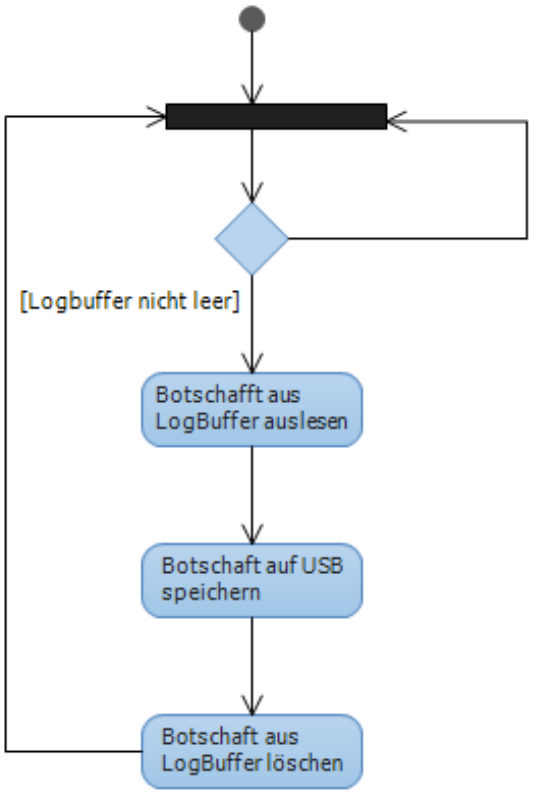

Abbildung 16 - Datenverarbeitung im Permanent-Modus

Das Aktivitätsdiagramm zeigt einen sich zyklisch wiederholenden Prozess, der bei Start einer Aufzeichnung bis zu deren Beendigung ausgeführt wird. Sobald ein Element im Logbuffer vorhanden ist, wird es aus diesem ausgelesen und in der entsprechenden Logdatei auf dem USB-Stick gespeichert. Anschließend wird das Element aus dem *LogBuffer* gelöscht, um den Speicherplatz im *LogBuffer* freizugeben.

Befindet sich der Datenlogger im Trigger-Modus, ist eine anderes Verfahren bei der Verarbeitung des *LogBuffer* erfoderlich. Nach dem Auslesen einer Botschaft auf dem LogBuffer ist zuerst eine Prüfung notwendig, ob diese Botschaft eine Trigger-Bedingung erfüllt. Erfüllt die Botschaft eine Trigger-Bedingung wird sie, sofort auf dem USB-Stick gespeichert. Anschließend ist der entsprechende Vorlauf und Nachlauf ebenfalls auf dem USB-Stick zu speichern. Das Schreiben des Nachlaufs lässt sich relativ einfach realisieren, indem die auf *Logtime* der Botschaft, welche den Trigger ausgelöst hat, zwei Sekunden addiert werden. Das Ergebnis stellt die Nachlaufzeit dar.

Da die Botschaften innerhalb des *Logbuffers* in zeitlicher Reihenfolge gespeichert werden, gehören alle Botschaften zum Nachlauf bei denen die *Logtime* kleiner oder gleich der zuvor berechneten Nachlaufzeit ist. Ein erneutes auslösen eines Triggers ist erst nach Ablauf der Nachlaufzeit möglich. Der Vorlauf eines Triggers befindet sich bei dessen Auslösen bereits vollständig im Vorlaufpuffer und könnte direkt auf dem USB-Stick gespeichert werden. Dieser Vorgang würde jedoch relativ viel Zeit in Anspruch nehmen. In dieser Zeit können keine weiteren Botschaften aus dem *Logbuffe*r abgearbeitet werden, was zu einem Überlauf führen wurde und somit einen Datenverlust zufolge hätte. Daher wird, ähnlich wie bei der Verarbeitung des *Logbuffers*, ein elementeweises Schreiben des Vorlaufpuffers vorgesehen. Analog des Vorgehens beim Nachlauf muss hierzu eine Speicherung der *Logtime* (Vorlaufzeit) der CAN-Botschaft erfolgen, welche den Trigger ausgelöst haben. Zum Vorlauf diese Trigger gehören alle Elemente deren *Logtime* kleiner der Vorlaufzeit und größer der Vorlaufzeit minus zwei Sekunden ist.

Handelt es sich bei einer Botschaft nicht um einen Trigger und sie gehört nicht zum Nachlauf eines solchen, ist diese Botschaft im Vorlaufpuffer abzulegen. Der Ablauf zur Verarbeitung von CAN-Botschaften im Trigger-Modus ist in dem in Abbildung 17 anhand eines Aktivitätsdiagramms dargestellt.

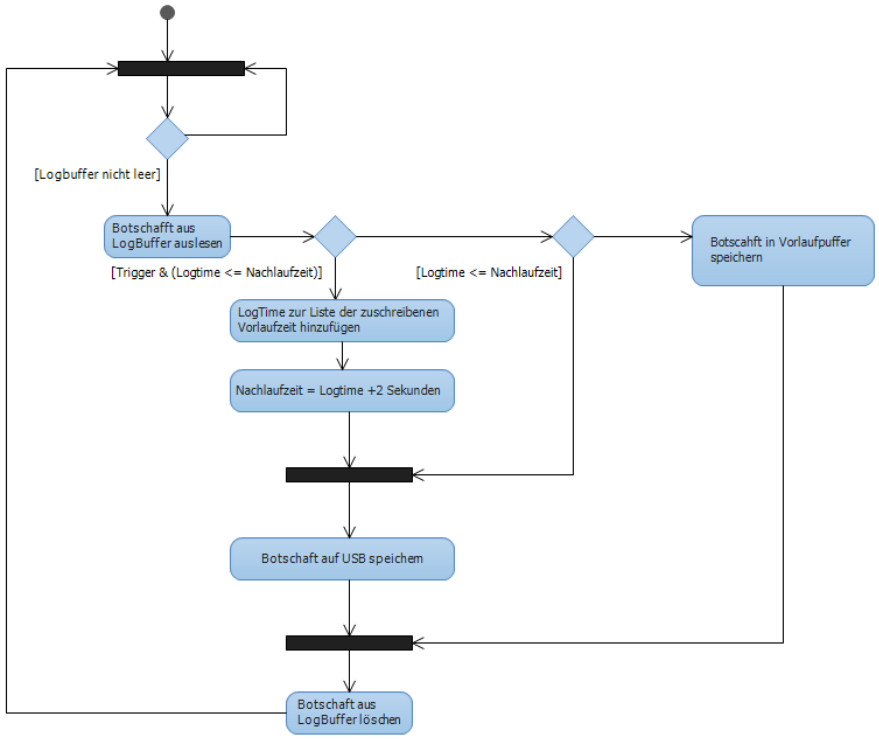

Abbildung 17 - Datenverarbeitung im Trigger-Modus

Beim Schreiben des Vorlaufs kann es zu Datenverlust kommen, wenn ein weiterer Trigger ausgelöst wird, bevor der Vorlauf des Vorigen komplett geschrieben wurde, da der neue Trigger die alte Vorlaufzeit überschreiben würde. Aus diesem Grund muss die Speicherung, wie in Abbildung 17, in einer Struktur erfolgen, welche mehrere Vorlaufzeiten aufnehmen kann z.B. einer Liste. Die einzelnen Vorlaufzeiten dieser Struktur sind dann der Reihe nach abzuarbeiten.

4.3.8 Speichern der Konfiguration in einem nichtflüchtigen Speicher

Damit der Datenlogger nach der Trennung der Spannungsversorgung nicht jedes Mal erneut konfiguriert werden muss, ist es sinnvoll die Konfiguration in einem nichtflüchtigen Speicher abzulegen. Dies ermöglicht das Laden zum Start der Betriebssoftware, sodass der Datenlogger sofort, ohne erneute Übertragung der Konfiguration verwendet werden kann.

Für die Speicherung der Konfiguration bietet das EA LPC4088 QuickStart Board zwei Möglichkeiten. Zum einem kann die Konfiguration auf dem USB-Stick gespeichert werden auf welchem auch das Speichern der aufgezeichneten Daten erfolgt. Diese Variante hat den Nachteil, dass der Anwender direkten Zugang auf die Konfiguration hat. Diese Tatsache birgt die Gefahr der Manipulation, der Konfiguration, oder der Vertauschung von USB-Sticks verschiedener Datenlogger. Hierdurch kann es zu Fehlern beim Laden der Konfiguration

durch die Betriebssoftware kommen. Diese Fehler können zu einem nicht gewünschten Verhalten oder zum Absturz der Betriebssoftware führen.

Die zweite und für diesen Fall bessere Variante ist die Speicherung der Konfiguration auf dem QSPI (Queued Serial Peripheral Interface)-Flash des EA LPC4088 QuickStart Boards. Auf diesen Speichern ist kein externer Zugriff mittels USB oder anderer Schnittstellen möglich. Dadurch kann eine Manipulation der Konfigurationsdaten weitestgehend ausgeschlossen werden. Der Zugriff auf diesen Speicher wird durch die Klasse *QSPIFileSystem* ermöglicht, welche in *EALib*- Bibliothek bereitgestellt wird [22].

4.3.9 Struktur der Betriebssoftware

Auf Basis der in den vorherigen Abschnitten gewonnenen Erkenntnisse wird die in der Abbildung 18 zu sehende Struktur für die Implementierung der Betriebssoftware vorgesehen. Anhand dieser Struktur wird die Funktionsweise der geplanten Implementierung erläutert.

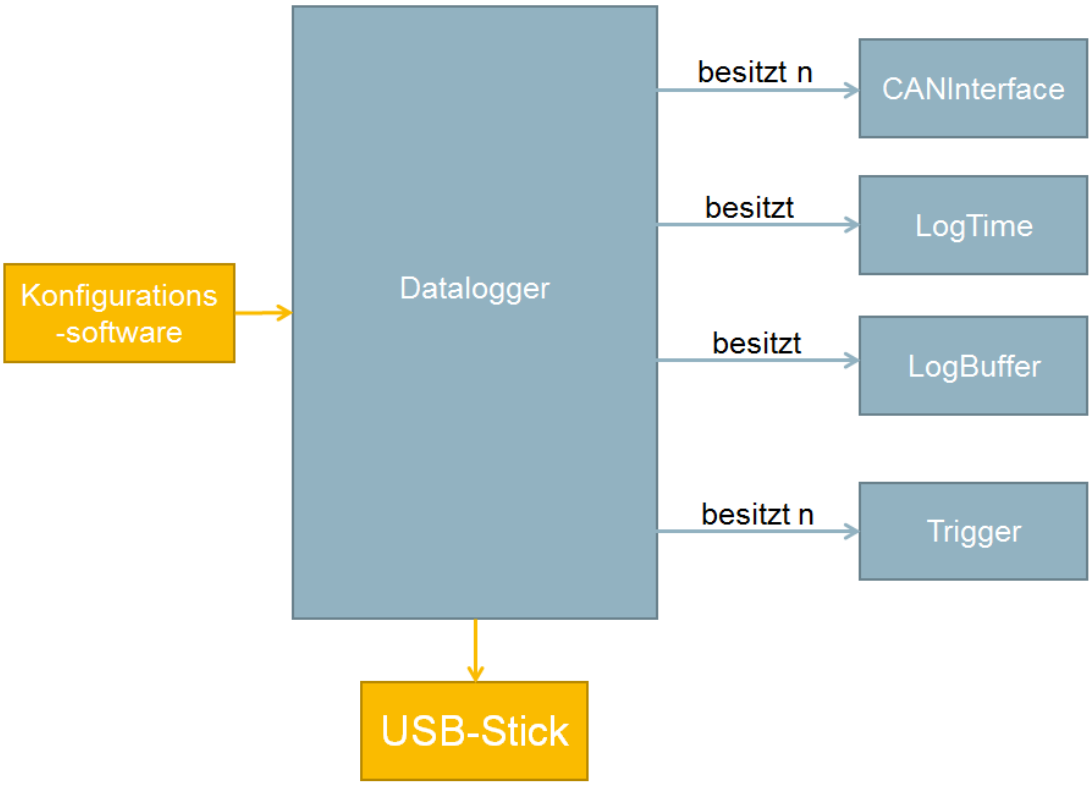

Abbildung 18 - Konzept Betriebssoftware

Die Klasse Datalogger bildet das Hauptelement der Software und besitzt die Objekte, der zuvor definierten, Klassen *CANInterface*, *LogTime*, *LogBuffer* und *Trigger*. Die Klasse Datalogger soll die Verarbeitung der von den *CANInterface*-Objekten im Zwischenspeicher (*LogBuffer*) abgelegten CAN-Botschaften realisieren. Über das *LogTime*-Objekt des Daten-

loggers können die *CANInterface*-Objekte die vergangene Zeit, seit Beginn der Aufzeichnung, abrufen. Die Objekte vom Typ *Trigger* des Datenloggers realisieren die Speicherung der vom Anwender konfigurierten Trigger-Bedingungen. Außerdem bildet die Klasse Datalogger die Schnittstelle zur Konfigurationssoftware und zum USB-Speichermedium. Die Funktionen zum Speichern der Daten auf dem USB-Stick sowie die Speicherung der Konfiguration auf dem QSPI-Flash, sollen ebenfalls durch die Klasse Datalogger realisiert werden.

4.4 Konfigurationssoftware-Konzept

In diesem Abschnitt wird das Konzept zur Umsetzung der Konfigurationssoftware erarbeitet. Zu Beginn des Konzeptes werden zuerst die Programmiersprache und das Grafik-Framework ausgewählt. Daraufhin erfolgt die Ermittlung der Use-Cases der Anwendung anhand der Anforderungen. Im Anschluss wird das Grundkonzept der Benutzeroberfläche erstellt und Konzepte zur Umsetzung der einzelnen Anwendungsfälle erarbeitet.

4.4.1 Auswahl Programmiersprache und Grafik-Framework

Microsoft bietet zur Erstellung von Benutzeroberflächen für Windows-Anwendungen die Auswahl zwischen zwei verschiedenen Grafik-Frameworks. Das erste Framework ist die WinForm-API (application programming interface), welches auch häufig als Windows Forms bezeichnet wird. Auf dieser Programmierschnittstelle basieren die meisten älteren Windows-Anwendungen. Obwohl Microsoft offiziell die Weiterentwicklung von Windows Forms eingestellt hat, ist die API auch weiterhin Bestandteil der Visual-Studio Entwicklungsumgebung und wird auch für die Umsetzung neuer Projekte genutzt. Die wesentlichen Nachteile von Windows Forms sind, dass keine strikte Trennung zwischen Benutzeroberfläche und Code vorhanden ist sowie die schlechte Individualisierbarkeit der Steuerelemente [23][24].

Die 2006 von Microsoft veröffentlichte Windows Presentation Foundation, kurz WPF, stellt die Alternative zur WinForm-API bei der Entwicklung von Windows-Anwendungen dar. Bei WPF erfolgt die Beschreibung der Benutzeroberfläche mit der an die Extensible Markup Language (XML) angelehnten eXtensible Application Markup Language (XAML). XAML ermöglicht eine strikte Trennung zwischen Benutzeroberfläche und Code [24]. Außerdem ermöglicht es XAML, viele Dinge deklarativ auszudrücken, für die in einer Windows Forms Anwendungen Code geschrieben werden müsste z.B. Animation und Datenbindungen von Steuerelementen. Die Trennung zwischen Benutzeroberfläche und Code ermöglicht eine Arbeitsteilung zwischen Designer und Softwareentwickler. Die Oberfläche kann z.B. vom Designer mit der Hilfe von Microsoft Blend bearbeitet werden, ohne dass sich dieser lange mit den Grundlagen der Softwareentwicklung und der Entwicklungsumgebung Visual Studio vertraut machen muss [23]. Das Aussehen einer WPF-Anwendung kann durch die Definition von sogenannten Styles festgelegt werden. Dies ermöglicht das einfache Anpassen

des Aussehens einer Anwendung, ohne den Programm- oder XAML-Code anzupassen. Einen weiteren Vorteil von WPF stellt die Vektorbasiertheit dar. Dieses verhindert ein verpixeln der Oberfläche und ermöglicht eine einfache Anpassung an verschiedene Monitorauflösungen [24].

Aufgrund seiner Vorteile wird für die Umsetzung der Konfigurationssoftware Windows Presentation Foundation als Grafik-Framework eingesetzt.

Der Programmcode für WPF-Anwendungen kann wahlweise in VB.NET oder C# geschrieben werden. Die beiden Sprachen unterscheiden sich weder hinsichtlich des Funktionsumfangs, noch hinsichtlich der Performance grundlegend voneinander. Aufgrund der höheren Verbreitung innerhalb des Unternehmens wird C# für die Programmierung der Konfigurationssoftware verwendet.

4.4.2 Ermittlung der Use-Cases

Aus den unter Kapitel 3 definierten Anforderungen ergeben sich die in Abbildung 19 als Anwendungsfalldiagramm dargestellten Use-Cases.

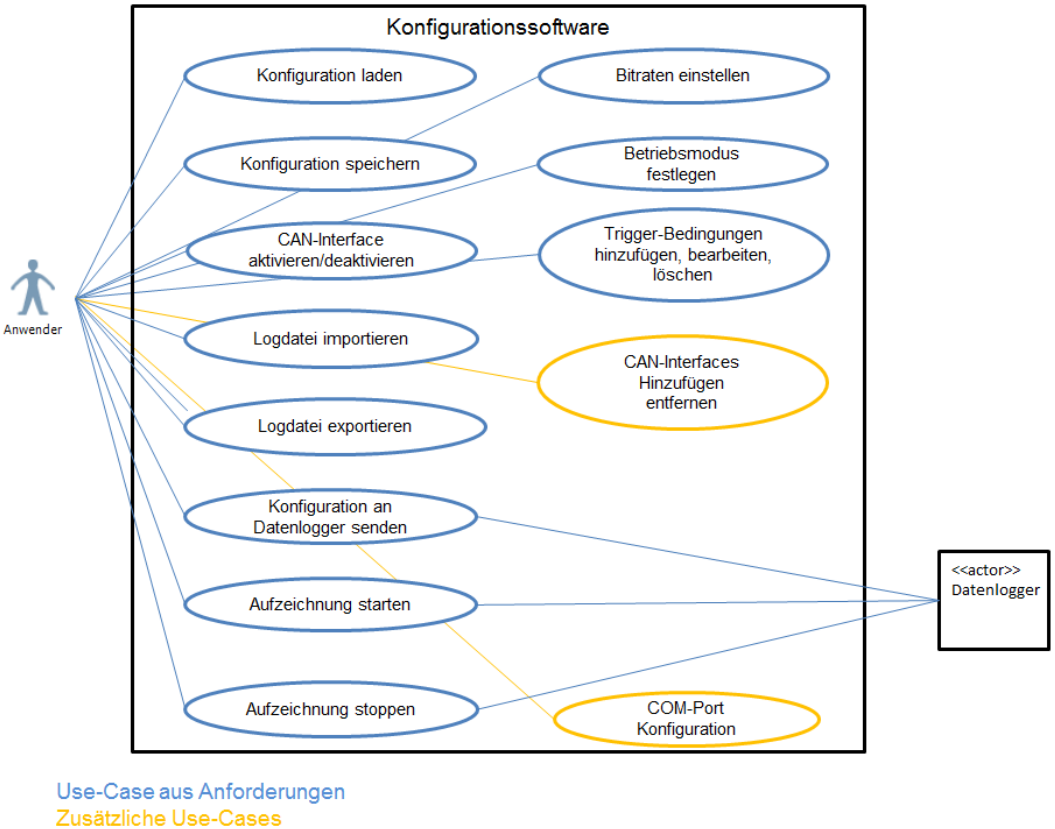

Abbildung 19 - Anwendungsfalldiagramm „Konfigurationssoftware“

Die in der Abbildung gelb umrandeten Use-Cases sind nicht in den Anforderungen enthalten. Die Gründe für die Aufnahme der beiden Use-Cases in die Konfigurationssoftware werden im Nachfolgenden genauer erläutert.

Der Use-Case *COM-Port Konfiguration* muss berücksichtig werden, da zur Kommunikation zwischen PC und Datenlogger die COM-Port-Schnittstelle festgelegt werden muss, an welcher der Datenlogger angeschlossen ist. Die Auswahl des COM-Ports könnte zwar möglicher automatisch durch die Software erfolgen, allerdings muss der Anwender spätestens beim zeitgleichen Anschluss von mehreren Datenloggern eine manuelle Auswahl treffen.

Das finale Produkt b.cube soll modular aufgebaut sein und kann über eine unterschiedliche Anzahl von CAN-Interfaces verfügen. Darum muss dem Anwender die Möglichkeit gegeben werden, seine Datenlogger-Konfiguration durch Hinzuzufügen oder Entfernen von CAN-Interfaces an die verwendete Hardware anzupassen.

4.4.3 Grundkonzept der Benutzeroberfläche

In diesem Abschnitt wird das Grundkonzept des Aufbaus der Benutzeroberfläche erläutert. Bei dem Konzeptentwurf der Benutzeroberfläche ist zu berücksichtigen, dass die Konfigurationssoftware später auch zur Konfiguration weiterer b.cube-Module genutzt werden soll. Diese geplante Ergänzung muss bereits bei dem ersten Entwurf berücksichtig werden. Einstellungen die mehrere Module betreffen, müssen zentral einstellbar sein. Ein Beispiel hierfür ist die Einstellung der Bitrate der CAN-Interfaces. Diese Einstellung betrifft sowohl das Datenlogger-Modul als auch das Gateway-Modul des b.cubes. Die Abbilddung 20 zeigt die Konzeptskizze der Benutzeroberfläche. Die Oberfläche besteht aus zwei Teilen. Zum einen den oben in blau dargestellten Menü und dem in grau dargestellten Bereich zur Darstellung des jeweiligen Inhalt.

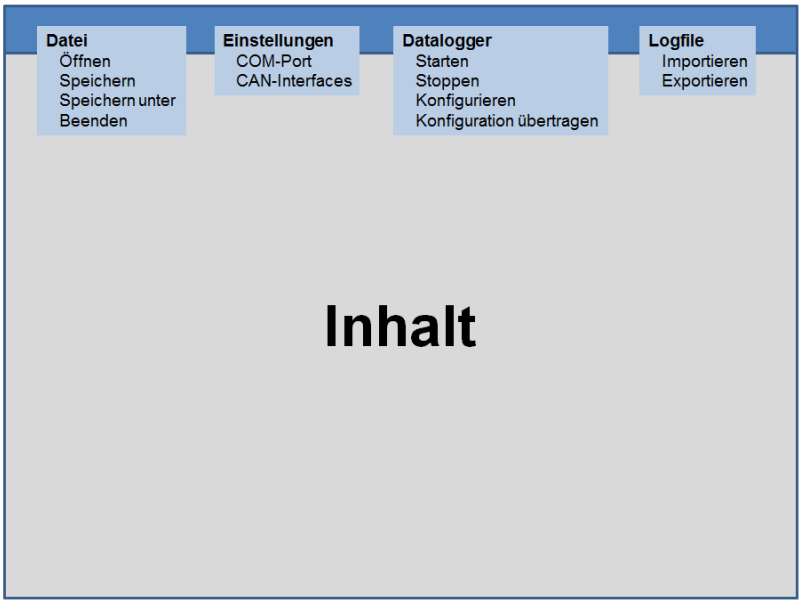

Abbildung 20 - Konzeptskizze Benutzeroberfläche der Konfigurationssoftware

Das in der Abbildung zu sehende Menü besteht aus den vier Hauptpunkten Datei, Einstellung, Datalogger und Logfile. Die vier Hauptkategorien und deren Unterpunkte werden aus den in Kaptiel 4.4.2 ermittelten Use-Cases abgeleitet. Die Tabelle 10 zeigt die Zuordnung der Use-Case zu den einzelnen Menüpunkten.

Tabelle 10 - Zuordnung von Uses-Cases zu Menüpunkten

Hauptpunkt	Unterpunkt	Zugeordneter Use-Case
Datei	Öffnen	Konfiguration laden
	Speichern	Konfiguration speichern
	Speichern unter	Konfiguration speichern
	Beenden	
Einstellung	COM-Port	COM-Port Konfiguration
	CAN-Interfaces	Bitrate einstellen
Datalogger	Starten	Aufzeichnung starten
	Stoppen	Aufzeichnung stoppen
	Konfigurieren	CAN-Interfaces aktivieren/deaktivieren, Triggerbedingungen, Betriebsmodus festlegen
	Konfiguration übertragen	Konfiguration an Datenlogger senden
Logfile	Importieren	Logdatei importieren
	Exportieren	Logdatei exportieren

Wie in der Konzeptskizze der Benutzeroberfläche zu sehen wird für die Anwendung ein einziges Fenster vorgesehen, dessen Inhalt, je nach ausgewähltem Menüpunkt, angepasst wird. Um die Anpassung des Fensters, je nach ausgewähltem Menüpunkt zu realisieren gibt es verschiedene Möglichkeiten. Zum einem können alle für die verschiedenen Anschichten benötigten Steuerelemente im Fenster platziert werden und je nach Bedarf ein- oder ausgeblendet werden. Diese Verfahren wird schnell unübersichtlich und hat außerdem den Nachteil, dass sämtlicher Code für die Interaktionslogik in einer Datei implementiert ist. Um die Übersichtlichkeit des Codes zu gewährleisten, wird das folgende Konzept zur Anpassung des Fensterinhalts für die Implementierung vorgesehen. Das Umsetzungskonzept

beruht auf den beiden WPF-Komponenten *ContentControl* und *Page.* Das *ContentConro*l ermöglicht die Darstellung einer Page innerhalb eines WPF-Fensters. Eine Page besteht wie ein WPF-Fenster aus zwei Dateien. In der einen Datei wird mittels XAML die Benutzeroberfläche definiert. Die andere Datei enthält die in C# codierte eigentliche Logik der Oberfläche. Diese Umsetzung ermöglicht eine Aufteilung der Benutzeroberfläche und deren Interaktionslogik auf mehrere Dateien. Je nach gewähltem Menüpunkt wird die entsprechende *Page* im *ContentControl* angezeigt. Durch die Aufteilung wird eine bessere Übersichtlichkeit innerhalb des Codes der Anwendung geschaffen. Für viele in der Abbildung 20 zu sehenden Menüpunkte ist keine Implementierung einer Page notwendig, da für diese Punkte keine Benutzeroberfläche notwendig ist, bzw. die Interaktion mit dem Benutzer mittels Dialogfenstern erfolgt (z.B. Datei öffnen).

Für die folgenden Menüpunkte wird die Erstellung einer eigenen WPF-Page vorgesehen:

- CAN-Interfaces
- Datenlogger konfigurieren
- Logfile importieren

Für die Konfiguration des COM-Ports wird die Implementierung der Benutzeroberfläche in einem Popupfenster vorgesehen.

4.4.4 Umsetzung der Use-Cases

In diesem Kapitel wird das Konzept zur Umsetzung der unter 4.4.2 ermittelten Use-Case der Konfigurationsanwendung entwickelt.

Der wesentlichen Use-Case der Anwendung ist die Erstellung und das Speichern einer Datenlogger-Konfiguration. Eine Konfiguration enthält Angaben zum Betriebsmodus, den einzelnen CAN-Interfaces und den vom Anwender definierten Triggern. Aus diesen Haupt-Use-Cases werden die in Abbildung 21 dargestellten Klassen *Config*, *CAN-Interfac*e und *Trigger* abgeleitet.

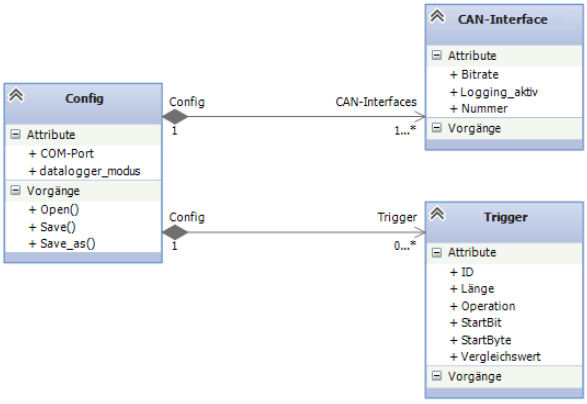

Abbildung 21 - Konzept Konfigurationssoftware Datenklassen

Die Implementierung dieser Klassen wird zur Speicherung der vom Anwender getroffenen Einstellungen vorgesehen. Das Hauptelement hierfür stellt die Klasse *Config* dar. Ein Objekt der Klasse enthält, neben den beiden Attributen zur Speicherung des eingestellten COM-Ports und des Betriebsmodus des Datenloggers, eine beliebige Anzahl von Objekten der Klasse CAN-Interface und Trigger. In diesen beiden Objekttypen sind die entsprechenden Attribute zur Einstellung eines *CAN*-Interfaces und zur Definition eines Triggers vorgesehen. Außerdem werden in der Klasse *Config* Methoden zur Realisierung der beiden Use-Cases „Konfiguration laden“ und „Konfiguration speichern“ vorgesehen.

Für die Speicherung der Konfiguration wird ein XML-Dateiformat vorgesehen. C# bietet umfassende Möglichkeiten zum Export und Import von Objekten aus einer XML-Datei, dieses erleichtert die Implementierung der Use-Cases „Konfiguration laden“ und „Konfiguration speichern“ erheblich.

Die Use-Cases „CAN-Interface hinzufügen/entfernen“ und „Bitrate einstellen“ werden wie im Grundkonzept der Benutzeroberfläche beschrieben, unter dem Menüpunkt „CAN-Interfaces“ zusammengefasst. Für diese drei Anwendungsfälle soll, wie im Grundkonzept festgelegt, eine gemeinsame Benutzeroberfläche in Form einer WPF-Page implementiert werden. Diese Oberfläche soll dem Anwender die folgenden Möglichkeiten bieten:

- Anzeige der CAN-Interfaces der Konfiguration
- Bearbeiten der Bitrate der angezeigten CAN-Interfaces
- CAN-Interfaces aus Konfiguration entfernen
- CAN-Interfaces zur Konfiguration hinzufügen

Für die Umsetzung der ersten drei Punkte bietet sich die Implementierung eines WPF-*DataGrids* an. In einem *DataGrid* können Objekte tabellarisch dargestellt werden. Jede Zeile kann Steuerelemente zur Bearbeitung des an sie gebundenen Objektes enthalten. Neben dem *DataGrid*, welches die Steuerelemente zur Anpassung der Bitrate und zum Löschen eines CAN-Interfaces enthält, wird ein Button zum Hinzufügen von CAN-Schnittstellen vorgesehen. Für das Hinzufügen eines Interfaces werden keine weiteren Steuerelemente zur Dateneingabe vorgesehen. Anstatt durch Benutzereingaben, sollen die Attribute des neuen CAN-Interfaces automatisch durch die Software festgelegt werden.

Die Bitrate wird standardmäßig auf 500 kbit/s festgelegt, da diese CAN-Bitrate am häufigsten in modernen Fahrzeugen verwendet wird. Die Nummerierung der CAN-Interfaces muss immer durchgängig sein und mit 1 beginnen. Darum ist bei der Implementierung der löschen Funktion darauf zu achten, dass eventuelle die Nummerierung der verbleidenden CAN-Interface angepasst werden muss.

Für die drei Anwendungsfälle „Aufzeichnung starten“, „Aufzeichnung stoppen“ und „Konfiguration an Datenlogger senden“ ist eine Kommunikation zwischen der Konfigurationssoftware und dem Mikrocontroller des Datenloggers notwendig. Die Kommunikation erfolgt mittels der unter Kapitel 4.2.3 festgelegten Kommandos. Die Kommunikation soll, wie in der

Schnittstellendefinition beschrieben, von der Konfigurationssoftware überwacht werden. Die Implementierung der Funktion, zur Übertragung der Kommandos und zur Überwachung der Übertragung wird in einer zentralen Funktion vorgesehen. Die zentrale Funktion realisiert außerdem die Ausgabe von Fehlermeldungen bei fehlerhafter Kommunikation. Die zentrale Implementierung der Kommunikationsfunktionalität bietet den Vorteil, dass die Logik für die Übertragung von Kommandos und Fehlererkennung nicht an mehreren verschiedenen Programmstellen umgesetzt werden muss. Dies erleichtert zukünftige Erweiterung der Kommunikationsschnittstelle. Abbildung 22 zeigt den zu implementierenden Grundumfang der Kommunikationsfunktion anhand eines Aktivitätsdiagramms.

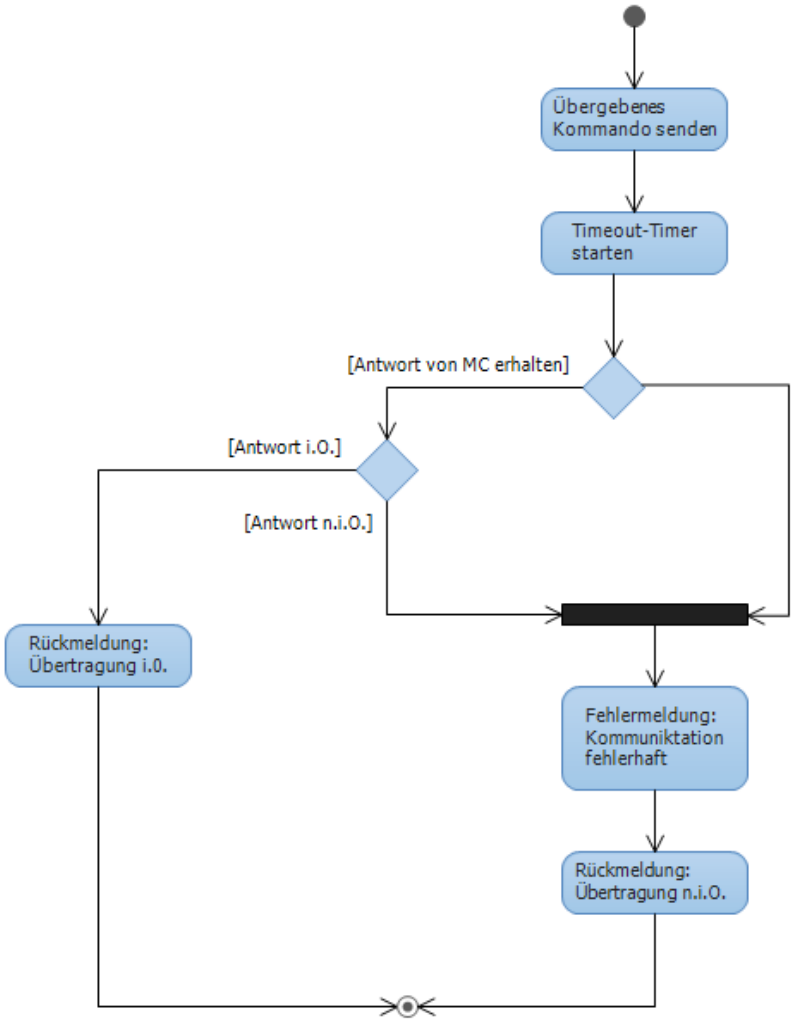

Abbildung 22 - Senden von Konfigurationskommandos

Zusätzlich zum Grundumfang ist in der Funktion eine Überprüfung vorzusehen, ob der Datenlogger hardwaremäßig über die in der Konfiguration festgelegten Anzahl von CAN-Interfaces verfügt. Sind mehr Interfaces in der Konfiguration vorhanden als tatsächlich physikalisch zur Verfügung stehen, müssen diese deaktiviert werden. Außerdem ist der Anwender über die Deaktivierung zu informieren.

Der unter 4.4.3 definierte Menüpunkt Datenlogger-Konfigurieren enthält den größten Funktionsumfang der Anwendung. Die Benutzeroberfläche für diesen Menüpunkt soll dem Anwender, neben Aktivierung oder Deaktivierung von CAN-Interfaces, zusätzlich die Möglichkeit zur Betriebsmodusauswahl und zur Triggerkonfiguration bieten. Für die Realisierung dieser Funktionalitäten sind viele Steuerelemente notwendig. Um die Benutzeroberfläche übersichtlich zu halten, wird für die Oberfläche eine Aufteilung der Funktionen auf zwei Registerkarten vorgesehen. Wobei eine Registerkarte die Funktionalität der Betriebsmodusauswahl und der Aktivierung und Deaktivierung von CAN-Interfaces enthalten soll, während in der anderen Registerkarte die Funktionalitäten zum Hinzufügen, Bearbeiten und Löschen von Trigger-Bedingungen zu implementieren werden.

Bei der Eingabemaske zum Bearbeiten und Hinzufügen von Triggerbedingungen wird eine Überprüfung der Nutzereingaben vorgesehen. Die durch den Anwender einzugebenden Attribute sind:

- ID der CAN-Botschaft
- Start-Byte des Signals
- Start-Bit des Signals
- Länge des Signals
- Vergleichsoperator (gleich, größer, kleiner, ungleich)
- Vergleichswert

Bei der eingegebenen ID ist zu prüfen, ob diese mit 29 Bit (maximal mögliche ID einer Botschaft) dargestellt werden kann. Der maximal zulässige Wert von Start Byte und Start Bit beträgt 7, da die Bits und Bytes von 0-7 durchnummeriert werden. Bei der Angabe der Länge des Signals ist darauf zu achten, dass die maximale Länge von 64 Bit in Abhängigkeit von Start Byte und Start Bit nicht überschritten wird. Die durch den Anwender eingegebene Länge ist korrekt, wenn die Bedingung $(StartBit + (8 * StartByte) + Signallänge) \leq 64$ erfüllt ist.

5 Implementierung

Bei der Implementierung der Betriebs- und Konfigurationssoftware erfolgt eine Aufteilung des Gesamtfunktionsumfanges auf mehrere Module. Die Modularisierung von Programmen stellt eine typisches Verfahren der objektorientierten Programmierung dar. Durch die Aufteilung eines Programms auf einzelne Module wird der Test der Software sowie eine spätere Erweiterung erleichtert. Ein weiterer Vorteil des Vorgehens ist die leichtere Wiederverwendbarkeit von einzelnen Programmteilen.

5.1 Betriebssoftware

Für die Umsetzung der Betriebssoftware werden, zusätzlich zu den im Konzept der Betriebssoftware im Kapitel 4.4 beschriebenen Klassen *Datalogge*r, *CANInterface*, *LogBuffer*, *LogTime* und *Trigger* die beiden Klassen *LogEntry* und *Triggerlist* implementiert. Die Klasse *LogEntry* dient der Definierung eines Datentyps für Elemente des *LogBuffers*. Die Realisierung der Verwaltung der einzelnen *Trigger* eines *Datalogge*r-Objekts erfolgt durch die Klasse *Triggerlist*. Die beiden Klassen dienen der Vereinfachung der Implementierung und stellen keine Änderung des erarbeiteten Konzepts der Betriebssoftware dar. Die Abbildung 23 zeigt die einzelnen Klassen der Betriebssoftware sowie die verwendeten Bibliotheken.

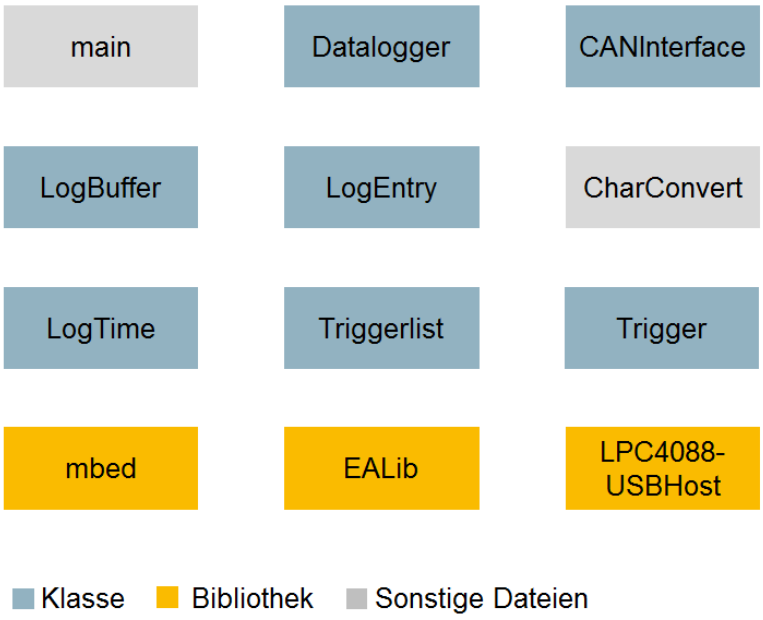

Abbildung 23 - Module der Betriebssoftware

Die in der Abbildung 23 dargestellte Datei *CharConvert* beinhaltet die im Konzept beschriebenen Funktionen zur Konvertierung von hexadezimalen Zeichen in Zahlenwerte. Durch die Auslagerung dieser Funktionen in eine separate Datei, wird die Wiederverwendung der Konvertierungsfunktion in anderen Projekten vereinfacht.

Bei der Implementierung der Betriebssoftware werden die Vorgaben des mbed-Frameworks zur Code-Formatierung sowie zur Benennung von Klassen, Methoden und Variablen berücksichtigt. Hierdurch wird eine Einheitlichkeit innerhalb des selbst implementierten Codes sowie zum Code der mbed-Bibliotheken geschaffen. Diese, auf der Website der mbed-Plattform, zu findenden Vorgaben enthalten zusätzlich eine Empfehlung zur Kommentierung des Quellcodes [25]. Diese Empfehlung wird ebenfalls bei Implementierung der Betriebssoftware umgesetzt und ermöglicht unter anderen die automatische Erstellung einer Klassendokumentation anhand der Quellcode-Kommentare. Zur Erstellung der Dokumentation wird das Programm Dogygen verwendet. Dieses analysiert die Kommentare der einzelnen Programmdateien und erstellt anschließend, anhand dieser, eine Dokumentation [26].

Zwischen den einzelnen Klassen der Betriebssoftware sowie den Klassen der verwendeten Bibliotheken bestehen Beziehungen. Diese Beziehungen sind in Abbildung 24 anhand eines vereinfachten Klassendiagramms dargestellt.

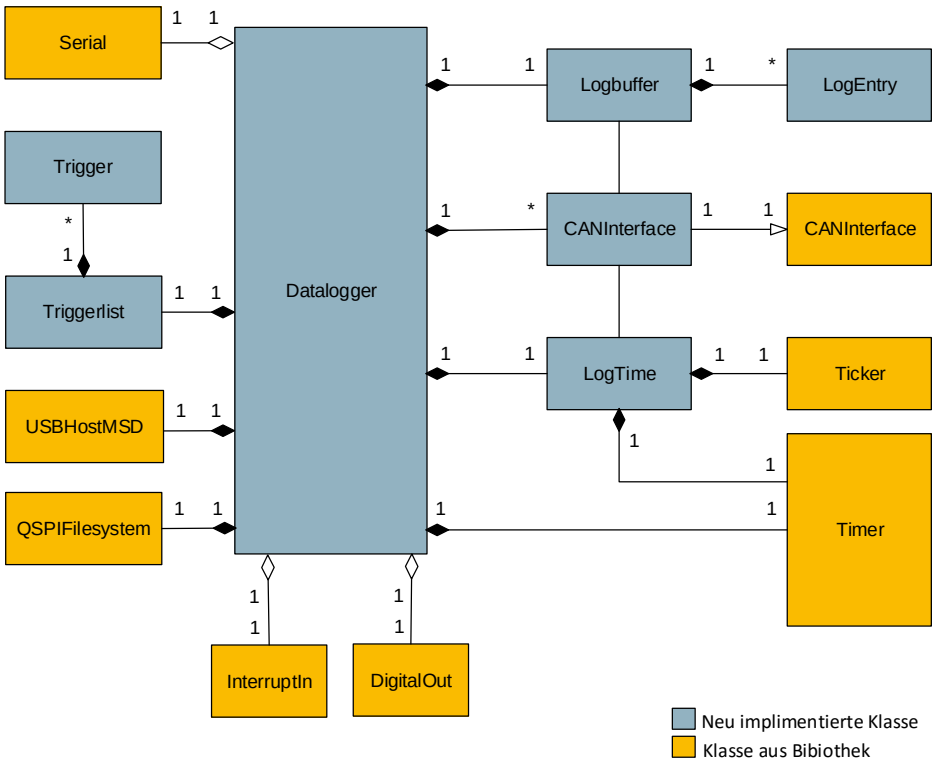

Abbildung 24 - Klassendiagramm Betriebssoftware

Im Nachfolgenden werden die Abhängigkeiten der Klassen sowie das Zusammenspiel derer Objekte zur Realisierung der einzelnen Funktionen der Betriebssoftware erläutert. Wie im Konzept der Betriebssoftware beschrieben, stellt die Klasse Datalogger das zentrale Element der Betriebssoftware dar. Ein Objekt der Klasse Datalogger besitzt eine beliebige Anzahl von Objekten vom Typ *CANInterface.* Diesen *CANInterface*-Objekten stellt es ein *LogBuffer*- und *LogTime*-Objekte bereit. Die *CANInteface*-Objekte rufen aus dem *LogTime* Objekt die vergangene Zeit seit Start des Logging-Vorgangs ab und speichern die empfangenen Botschaften im bereitgestellten LogBuffer, in Form eines Objektes vom Typ *LogEntry*, ab. Die Implementierung der Klasse *LogTime* erfolgt, wie im Konzept unter 4.3.2 beschrieben mit Hilfe der beiden Klassen Ticker und Timer der mbed Bibliothek. Die Verarbeitung der durch die *CANinterface*-Objekte im *LogBuffer* abgelegten *LogEntry*-Objekte erfolgt durch das Datalogger-Objekt. Während der Verarbeitung erfolgt bei aktivem Trigger-Modus eine Überprüfung auf eine erfüllte Trigger-Bedingung mit Hilfe des *Triggerlis*t-Objektes eines Datenloggers. Dieses Objekt übernimmt die Verwaltung der durch den Anwender konfigurierten Tigger-Bedingungen, welche in Form von Objekten der Klasse *Trigger* gespeichert werden. Die Umsetzung des, im Konzept unter 4.4.5 beschriebenen Vorlaufpuffers erfolgt in der Klasse Datalogger. Ein Objekt der Klasse *Datalogger* verfügt außerdem über ein Objekt der Klasse *USBHostMSD*, durch welches die Anbindung des USB-Sticks realisiert wird. Die Speicherung der Konfiguration des Datenloggers auf dem QSPI-Flashs des EA LPC4088 Quickstart Board wird ebenfalls durch die Klasse Datenlogger, mit Hilfe eines Objekts der Klasse *QSPIFilesystem* realisiert, welches den Zugriff auf diesen Speicher ermöglicht. Über das *DigitalOut*-Objekt eines Datalogger-Objektes erfolgt die Ansteuerung der Betriebs-LED des Datenloggers. Diese LED dient zur Signalisierung einer aktiven Aufzeichnung. Das Objekt des Typs *InterruptIn* eines Datalogger-Objektes realisiert das Auslösen eines Triggers mittels Tastendruck.

Eine detaillierte Dokumentation der Attribute und Methoden der Klassen der Betriebssoftware, in Form einer Dogygen-Dokumentation, ist auf der CD zu finden, welche dieser Diplomarbeit beiliegt.

Bei der Implementierung der Betriebssoftware sind einige Probleme aufgetreten. Im Nachfolgenden werden ausgewählte Probleme und deren Lösung beschrieben. Zwischen den beiden Klassen *Datalogger* und *CANInterface* besteht eine starke Abhängigkeit. Die Erstellung eines *Datalogger*-Objektes bedingt ein *CANInterface*-Objekt und umgekehrt. Damit in C++ in einer Klasse der Typ einer anderen Klasse bekannt ist, erfolgt normalerweise ein Include des entsprechenden Header-Files. Dieses Vorgehen ist bei dieser starken Abhängigkeit nicht möglich und führt zu einem Fehler beim Kompilieren der Software. Damit das Kompilieren der Software möglich, ist wurde in der Klasse Datalogger der Include der Klasse *CANInterface* entfernt. Damit der Typ der *CANInterface* trotzdem in der Klasse *Datalogge*r verwendet werden kann wird er mittels Prototypen (Class *CANInterace*;) bekannt gegeben.

Ein weiteres Problem ist bei der ursprünglich geplanten Realisierung der Verarbeitungsfunktion der Klasse Datalogger aufgetreten. Diese Funktion sollte in einem Thread innerhalb der Klasse Datalogger ausgeführt werden. Das mbed-Framework unterstützt zwar grundsätzlich das Anlegen von Threads allerdings ist die Erstellung eines Threads innerhalb von Klassen nicht möglich. Aus diesem Grund erfolgt die Implementierung der Verarbeitungsfunktion als öffentliche Methode der Datalogger-Klasse, welche innerhalb einer WHILE-Schleife in der *Main* der Betriebssoftware ausgeführt wird.

Außerdem kam es bei ersten Tests bei längeren Aufzeichnungen zu einem Absturz des Programms. Die Ursache hierfür war der Pufferspeicher für den Vorlauf im Triggermodus. Durch diesen wurde ein RAM-Überlauf verursacht, obwohl die Anzahl der Elemente im Puffer sich unterhalb der im Konzept festgelegten Obergrenze befand. Es stellte sich heraus, dass der 32 MByte SRAM des Mikrocontrollers zuerst initialisiert werden muss, da ansonsten nur 512 Kbyte RAM zur Verfügung stehen.

Ein weiterer Fehler trat bei der Einstellung der Bitrate einer CAN-Schnittstelle auf. Die eingestellte Bitrate wurde nicht übernommen. Bei der Analyse des Problems stellte sich heraus, dass diese Probleme von der verwendeten Bibliothek zur USB-Stick Anbindung verursacht wird. Die Verwendung der Bibliothek verhindert die korrekte Initialisierung der CAN-Schnittstellen. Das Problem wurde gelöst, indem die Initialisierungswerte beim Start des Programmes direkt in die entsprechenden Register geschrieben werden. Um das Problem nachhaltig zu lösen, muss die fehlerhafte Bibliothek ersetzt oder deren Funktionsumfang selbst implementiert werden.

Die Ersetzung der fehlerhaften Bibliothek sowie die Implementierung der Funktionalitäten zur Aufteilung von Aufzeichnungen auf mehre Dateien und die Kapazitätsprüfung des Speichermediums, werden aus zeitlichen Gründen für eine spätere Version der Betriebssoftware vorgesehen.

5.2 Konfigurationssoftware

Bei der Implementierung der Konfigurationssoftware werden zur Kommentierung des Quellcodes XML-Dokumentationskommentare verwendet. Die Kommentarform ermöglicht die automatische Erstellung einer Dokumentation der Software. Die Dokumentation wird von Visual Studio in Form einer XML-Datei erstellt [27]. Um diese Dokumentation in lesbare eine Form zu konvertieren, wird das Programm Sandcastle verwendet. Sandcastle ermöglicht die Konvertierung der XML-Dokumentation in eine Compiled HTML Help (cmh) [28].

Die einzelnen Komponenten der Software lassen sich, wie in Abbildung 25 dargestellt, in die zwei Kategorien Benutzeroberfläche sowie Funktions-und Datenklassen unterteilen. Die einzelnen Komponenten dieser beiden Kategorien werden in den nachfolgenden Abschnitten erläutert.

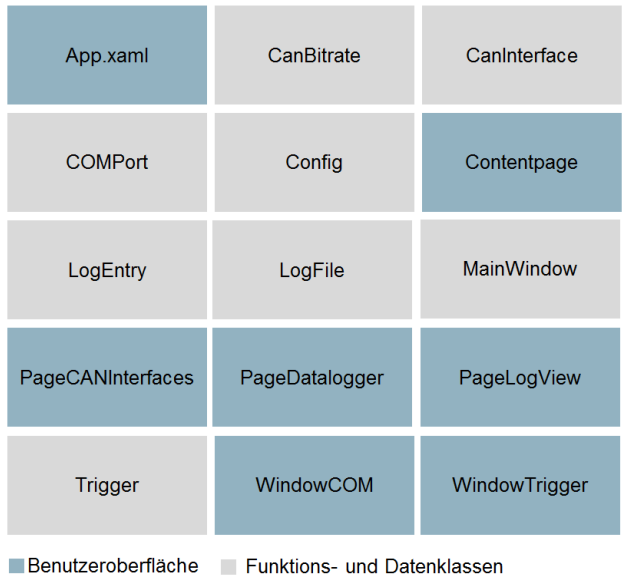

Abbildung 25 - Komponenten der Konfigurationssoftware

5.2.1 MainWindow

Bei dem *MainWindow* handelt es sich um das zentrale Element der Benutzeroberfläche. Dieses Fenster (Window) wird beim Start der Anwendung geöffnet und stellt die Hauptschnittstelle zwischen Benutzer und Anwendung dar. Das *MainWindow* wird, wie alle Fenster in WPF, durch zwei Dateien, *MainWindow*.xaml und *MainWindow*.xaml.cs, definiert. Die Festlegung des Aussehens und die Anordnung der Steuerelemente erfolgt in der MainWindow.xaml. Die Logik für die Interaktion mit dem Anwender wird in der *MainWindow*.xaml.cs definiert. Die Abbildung 26 zeigt das Layout des *MainWindows*, welches in der *MainWindow*.xaml mittels XAML beschrieben wird. In der Abbildung 26 sind die Hauptsteuerelemente des Fensters markiert.

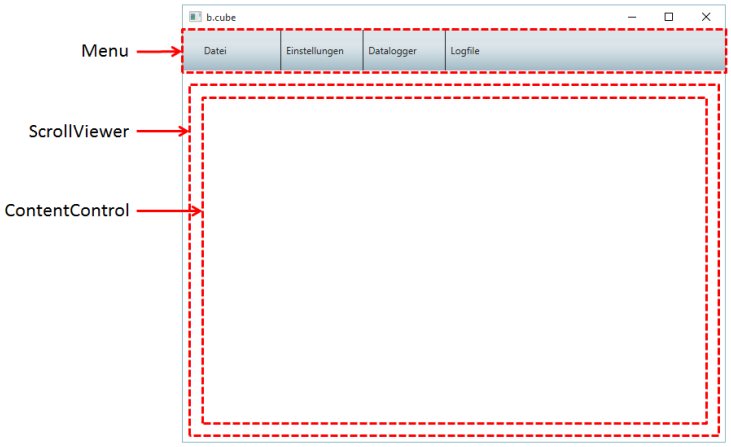

Abbildung 26 - MainWindow

Das Menü enthält die im Konzept der Konfigurationssoftware definierten Unterpunkte. Das *ScrollViewer*-Steuerelement verhindert bei einer zu kleinen Fenstergröße das Abschneiden des im *MainWindow* dargestellten Inhalts, indem es einen entsprechenden Scrollbalken einfügt. In dem *ContentControl* werden, wie im Konzeptteil beschrieben, die verschiedenen Inhaltsseiten (Pages) der Konfigurationssoftware dargestellt.

5.2.2 App.xaml

In der Datei *App.xaml* werden sogenannte Styles für die Steuerelemente der Anwendung definiert. Ein Style beschreibt z.B. die Hintergrundfarbe, die Abmessung eines Elementes, oder auch den Mouseovereffekt eines Elementes. Durch das Definieren eines Styles für ein Steuerelement müssen die Eigenschaften eines Steuerelementes nur einmalig an zentraler Stelle festgelegt werden. Dieses erleichtert die Implementierung eines einheitlichen Aussehens aller Steuerelemente eines Typs. Außerdem wird die spätere Anpassung des Aussehens der Anwendung wesentlich vereinfacht, da nur einmalig an zentraler Stelle geändert werden muss.

5.2.3 ContentPage

Bei *ContentPage* handelt es sich um eine abstrakte Klasse, welche von *System.Windows.Controls.Page* abgeleitet wird. Sie enthält die abstrakte Methode *refresh*. Diese Klasse wird implementiert, damit jede der Inhaltsseiten (Pages), welche von *ContentPage* abgeleitet wird, automatisch um eine Methode zur Aktualisierung des Inhaltes erweitert wird. Welcher Inhalt der einzelnen Pages genau durch die Methode *refresh* aktualisiert wird**,** wird in der Codebehind Datei der einzelnen Pages festgelegt. Durch die Ableitung der einzelnen Pages von der abstrakten Klasse ContentPage wird lediglich sichergestellt, dass jede Inhaltsseite eine solche Methode enthält. Die Implementierung dieser Klasse ermöglicht es, dass der Inhalt des MainWindows, unabhängig von der angezeigten Inhaltsseite mittels der Methode *refresh* aktualisiert werden kann.

Die Abbildung 27 zeigt eine Übersicht aller Inhaltsseiten, welche von der Klasse *ContentPage* abgeleitet werden.

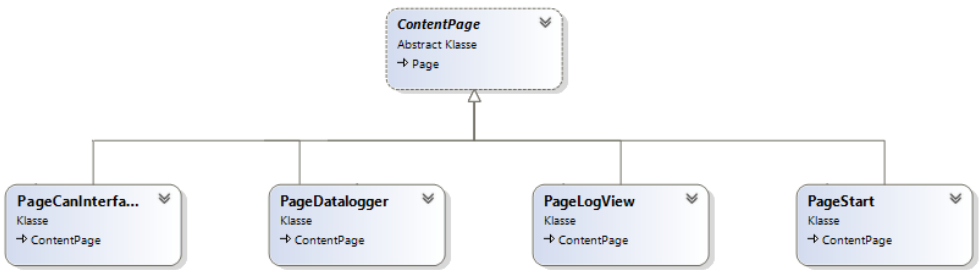

Abbildung 27 - Übersicht der von ContentPage abgeleiteten Klassen

5.2.4 WindowCOM

Die Abbildung 28 zeigt das Fenster *WindowCOM*. Dieses Fenster ermöglicht es den Anwender den COM-Port auszuwählen, überwelchen der Datenlogger mit dem PC verbunden ist.

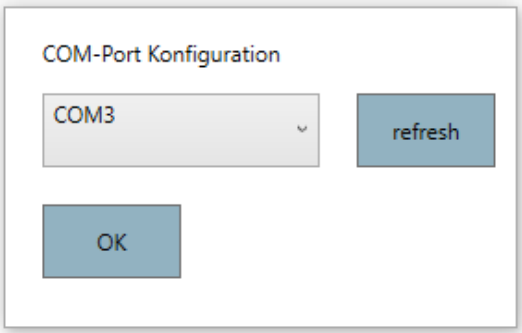

Abbildung 28 - WindowCOM

Die Klasse enthält neben dem Konstruktor und den beiden Methoden für die Reaktion auf die von den Buttons aufgelösten Ereignissen, die Methode *getDevices*. Die Methode getDevices realisiert das Abrufen einer Liste der aktuell am PC verwendeten COM-Ports (Geräte angeschlossen) und legt diese Liste anschließend als Quelle für die Combobox (Dropdown) fest. Der Abruf der Liste der COM-Ports erfolgt mittels der vom .NET-Framework bereitgestellten Funktion *SerialPort.GetPortNames.* Der interne Ablauf der Methode *getDevices* ist als Ablaufdiagramm in der Abbildung 29 dargestellt.

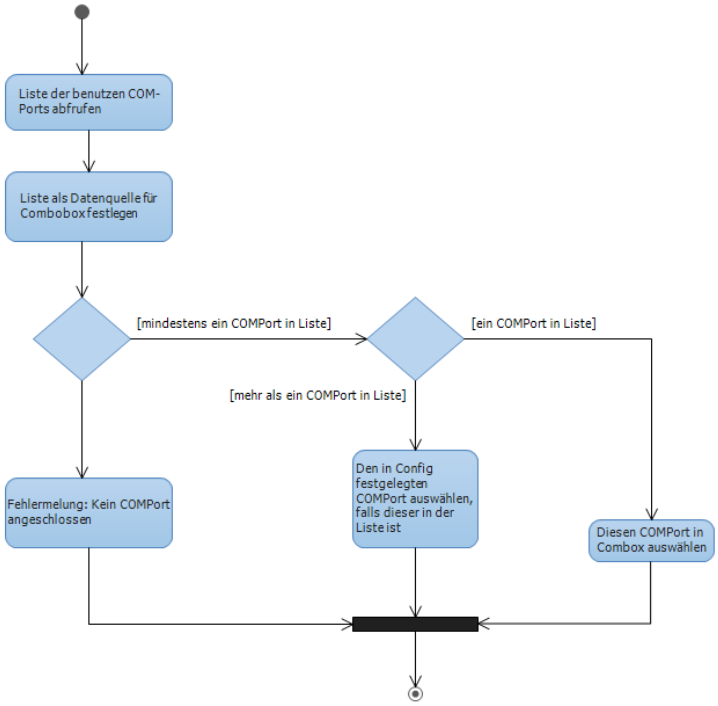

Abbildung 29 - Ablaufdiagramm getDevices

5.2.5 PageCANInterfaces

Die in der Abbildung 30 dargestellte Page *CANInterfaces* stellt die Oberfläche und die Logik für die Konfiguration der CAN-Schnittstellen bereit.

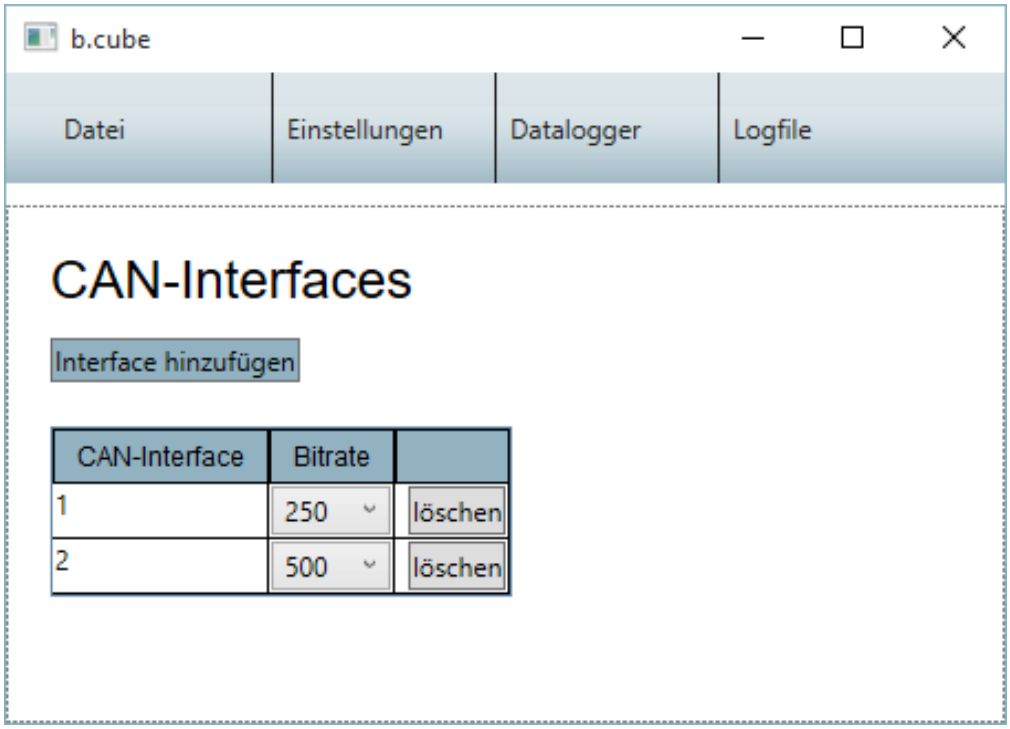

Abbildung 30 - PageCANInterfaces

Wie im Konzept der Betriebssoftware beschrieben, beinhaltet die Page ein DataGrid zur Anzeige der vorhandenen CAN-Schnittstellen. Außerdem ermöglicht das DataGrid das Einstellen der Bitrate des entsprechenden Interfaces mittels einer Dropdown-Liste und das Löschen des Interfaces. Die Benutzeroberfläche Page CANInterfaces enthält außerdem einen Button zum Hinzufügen eines neuen CAN-Interfaces. Wie im Konzept festgelegt, sind für das Hinzufügen einer neuen CAN-Schnittstelle keine weiteren Angaben notwendig. Das Festlegen notwendiger Attribute erfolgt automatisch durch die Software.

5.2.6 PageDatalogger

In der *PageDatalogger* ist die Benutzeroberfläche und die Interaktionslogik zur Konfiguration des Datenloggers enthalten. Diese Page ermöglicht es dem Benutzer die Aufzeichnung von bestimmten CAN-Interfaces zu aktivieren oder zu deaktivieren, den Betriebsmodus zu wechseln, das automatische Starten und Stoppen von Aufzeichnungen zu aktivieren und Trigger-Bedingungen zu konfigurieren. Wie im Konzept beschrieben, wird die Darstellung in zwei Bereiche aufgeteilt. Die Aufteilung der Page erfolgt durch Verwendung eines TabControls. Dieses ermöglicht verschiedene Registerkarten zu Erstellen und mittels eines Menüs durch diese zu navigieren.

Im ersten, in Abbildung 31 dargestellten, Bereich befinden sich die Steuerelemente zur Aktivierung und Deaktivierung der einzelnen Interfaces und zur Festlegung des Betriebsmodus sowie zur Aktivierung des automatischen Startens und Stoppen der Aufzeichnung (Automodus).

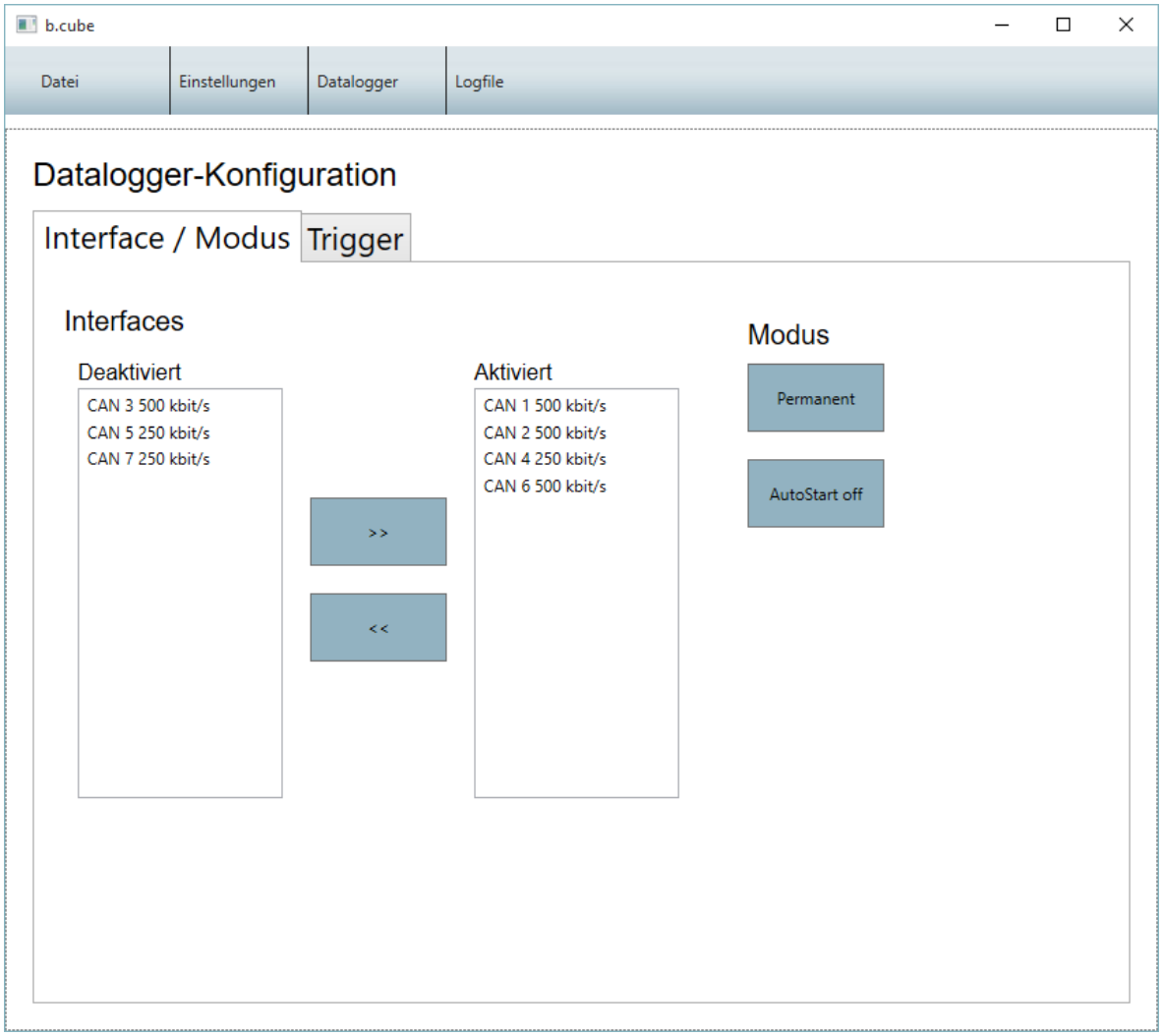

Abbildung 31 - Benutzeroberfläche Interface- und Betriebsmodus-Konfiguration

In der zweiten, in Abbildung 32, darstellen Registerkarte sind die Einstellungen für die Trigger-Bedingungen zu finden. Über einen Dialog können Trigger hinzugefügt und bearbeitet werden.

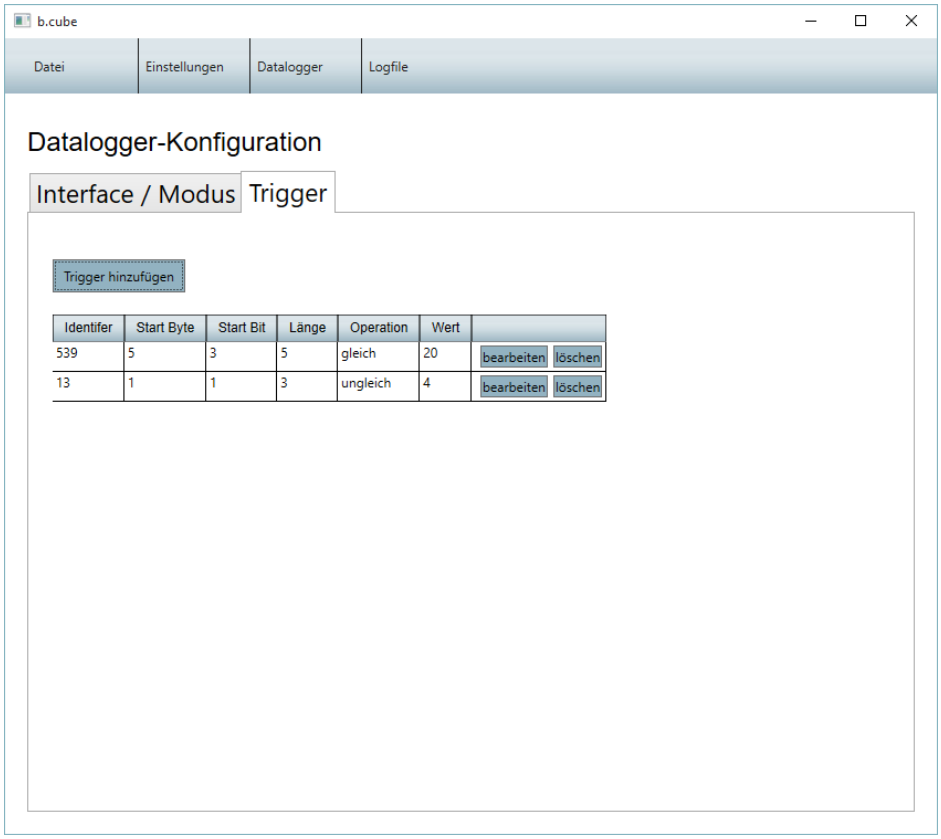

Abbildung 32 - Benutzeroberfläche Trigger-Konfiguration

Die Funktionsweise ist in die Klasse *WindowTrigger* ausgelagert, welche auch die Überprüfung der Eingaben vornimmt. Die Benutzeroberfläche des Fensters *WindowTrigger* ist in der Abbildung 33 dargestellt.

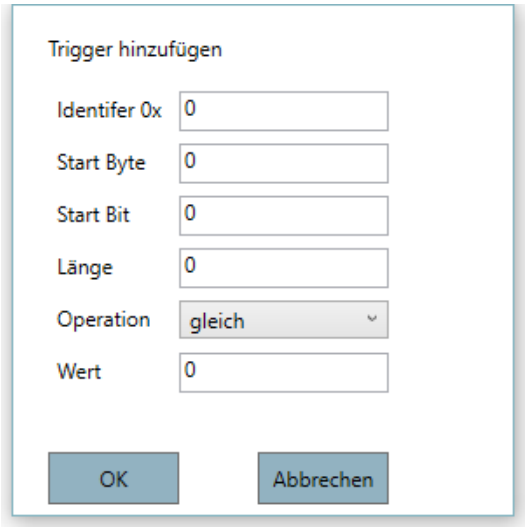

Abbildung 33 - WindowTrigger

5.2.7 Konfiguration des Dataloggers

Die, über die Benutzeroberfläche eingestellten, Daten werden in einem Objekt der Klasse *Config* gespeichert. Außerdem enthält diese Klasse die Funktionalität zum Import und Export einer Konfiguration in eine XML-Datei. Zur Darstellung des Datenmodells der Konfiguration werden zusätzlich die Klassen *CanInterface*, *CanBitrate* und *Trigger* verwendet.

In Abbildung 34 sind die Beziehungen zwischen den Klassen *Config*, *CANInterface*, *CanBitrate* und *Trigger* dargestellt.

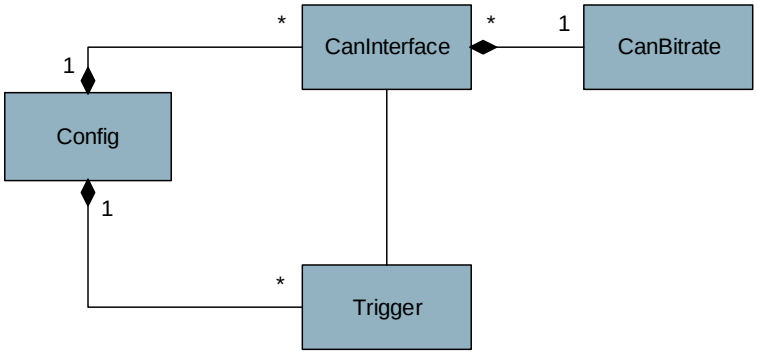

Abbildung 34 - Klassenbeziehungen der Konfigurationssoftware

Die, in einem Objekt vom Typ *Config* gespeicherte, Konfiguration eines Datenloggers besteht, wie in der obigen Abbildung dargestellt, aus einer beliebigen Anzahl von Objekten vom Typ *CANInterface* und *Trigger*. Jedes *CANInterface*-Objekt ist mit einem Objekt vom Typ *CanBitrate* verknüpft.

Die Übertragung der Konfiguration an den Datenlogger erfolgt mithilfe der Klasse *COMPort*. Die Klasse realisiert die Datenübertragung zum Datenlogger und überwacht die Verbindung.

5.2.8 Import und Export von LogDateien

Der Import und Export der vom Datenlogger aufgezeichneten Daten wird durch die Klasse *LogFile* realisiert. Die einzelnen Datensätze werden durch Objekte der Klasse *LogEntry* repräsentiert. Beim Import wird die Logdatei byteweise in Objekte vom Typ *LogEntry* verarbeitet. Beim Auftreten von Fehlern wird das Einlesen abgebrochen und eine entsprechende Fehlermeldung an den Anwender ausgegeben. Nach dem erfolgreichen Import werden die importierten Datensätze in der PageLogView angezeigt (Abbildung 35).

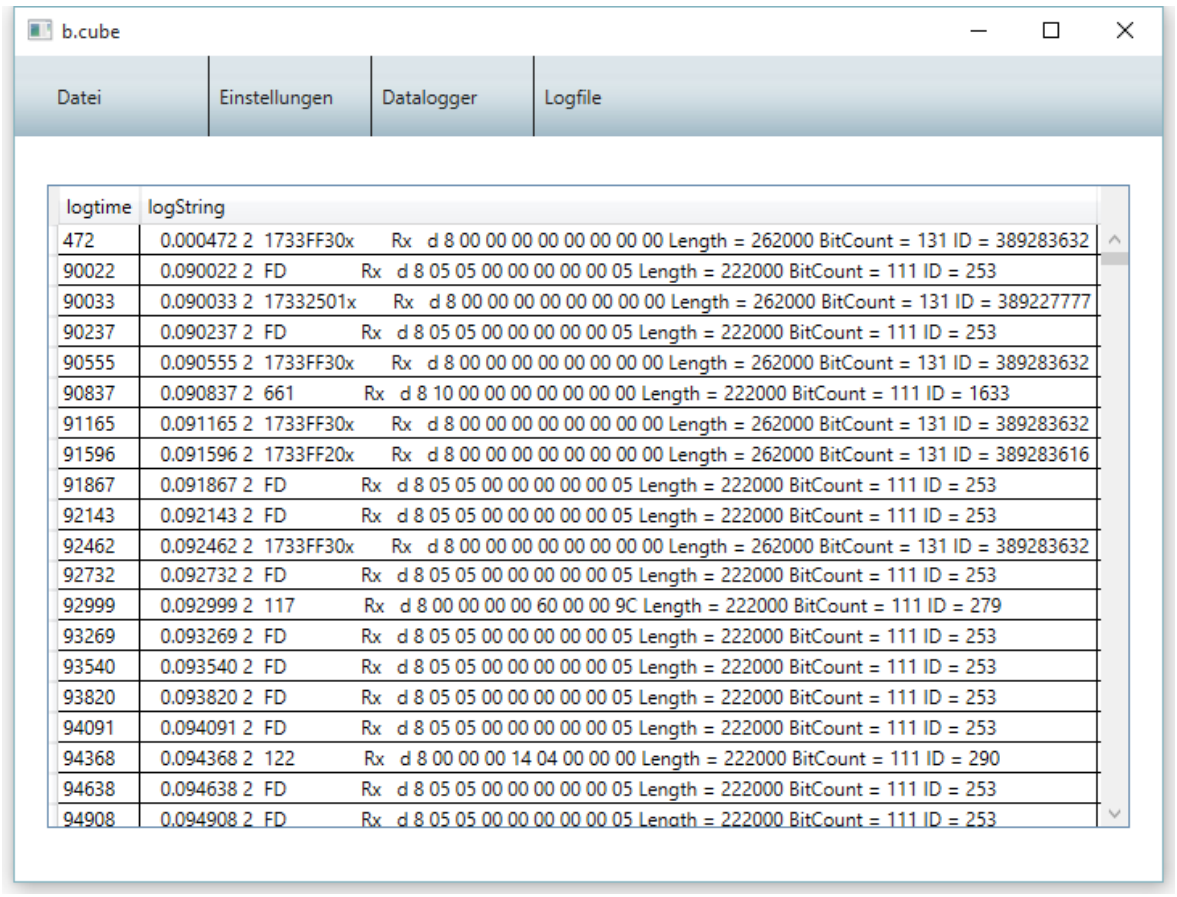

Abbildung 35 - PageLogView

Beim Export werden die *LogEntrys* ins ASC-Format konvertiert und in einer Datei gespeichert.

6 Softwareverifikation

Dieses Kapitel beschreibt die Verifikation der implementierten Betriebs- und Konfigurationssoftware. Durch die Verifikation wird belegt ob die Software die an Sie gestellten Anforderungen (Kapitel 2) erfüllt.

6.1 Testumgebung

Dieser Abschnitt beschreibt den Testaufbau sowie die für die Durchführung verwendet Hardware und Software. Die Abbildung 36 zeigt den Aufbau der Testumgebung zur Verifikation der Betriebs- und Konfigurationssoftware.

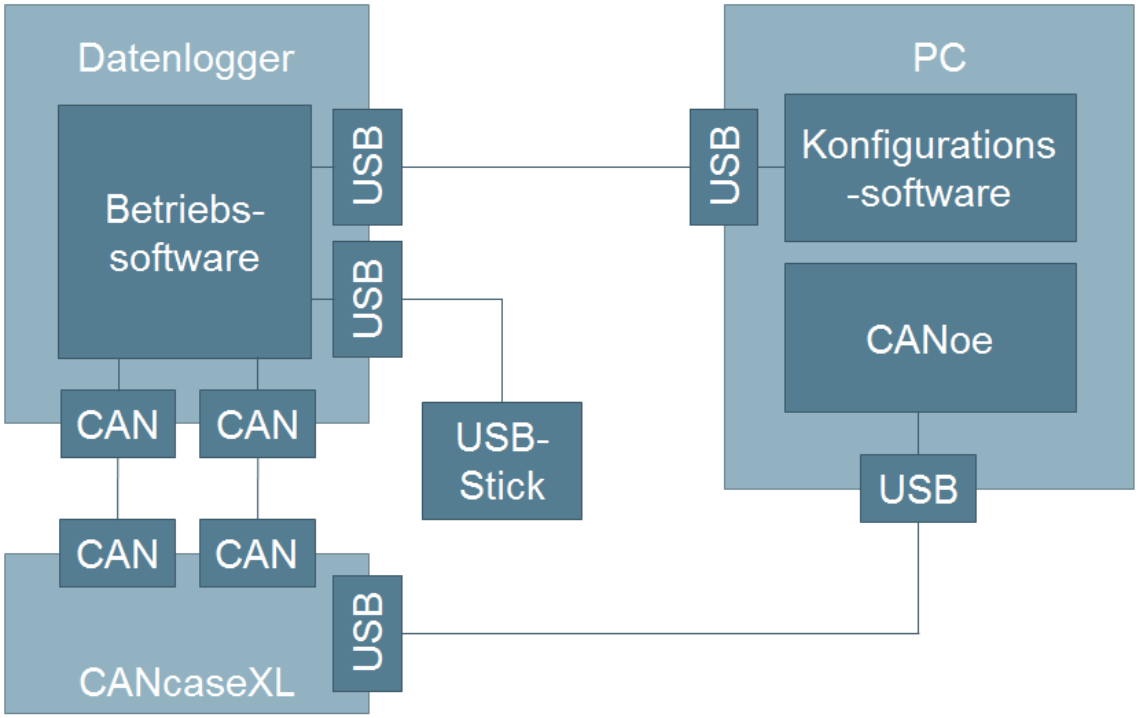

Abbildung 36 - Testaufbau

Zusätzlich zu den zur Ausführung der Betriebs- und Konfigurationssoftware benötigten Hardware wird für den Test eine CANcaseXL und die Software CANoe verwendet. Bei dem CANcaseXL handelt es sich um eine USB-Interface der Firma Vector Informatik. Dieses Interface ermöglicht das Versenden und Empfangen von CAN-Botschaften mittels der PC-Software CANoe [10][29]. Bei dem nachfolgenden Test wird CANoe verwendet um CAN-Botschaften an die CAN-Schnittstellen des Datenloggers zu senden.

6.2 Testfalldefinition

In diesem Abschnitt werden die Testfälle zur Überprüfung der Anforderung definiert. Für jeden Testfall werden Vorbedingungen, auszuführende Testschritte und das erwartete Ergebnis definiert. Zusätzlich wird für jeden Testfall angegeben, auf welche Anforderung aus Kapitel 3 er sich bezieht.

Tabelle 11 - Testfall T01

Nummer	T01
Titel	Einstellung der Bitraten
Anforderungsbezug	A02
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen
auszuführende Testschritte	1. Mittels Konfigurationssoftware Bitraten der CAN-Interfaces einstellen 2. Konfiguration an Datenlogger senden 3. In CANoe den Bitraten-Scanner ausführen
erwartetes Ergebnis	Die von CANoe ermittelte Bitrate entspricht der eingestellten

Tabelle 12 - Testfall T02

Nummer	T02
Titel	Betriebsmodus Permanent
Anforderungsbezug	A01
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen
auszuführende Testschritte	1. Datenlogger Konfigurieren (2 CAN-Schnittstellen 500 kbits/s, Betriebsmodus: Permanent, kein AutoStart) 2. Aufzeichnung über Konfigurationssoftware starten 3. CAN-Botschaften mittels CANoe senden 4. Senden der Botschaften beenden 5. Aufzeichnung Stoppen 6. Aufzeichnung ins ASC-Format konvertieren
erwartetes Ergebnis	Die konvertierte Aufzeichnung enthält alle Botschaften die per CANoe gesendet wurden. Die Botschaften wurden in der richtigen Reihenfolge und im richtigen zeitlichen Abstand aufgezeichnet.

Tabelle 13 - Testfall T03

Nummer	T03
Titel	CAN-Schnittstelle deaktivieren
Anforderungsbezug	A02
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen
auszuführende Testschritte	1. Dieselbe Datenlogger-Konfiguration wie in Testfall T02 verwenden, jedoch eine CAN-Schnittstelle deaktivieren 2. Aufzeichnung über Konfigurationssoftware starten 3. CAN-Botschaften an beide CAN-Schnittstellen des Datenloggers senden 4. Senden der Botschaften beenden 5. Aufzeichnung Stoppen 6. Aufzeichnung ins ASC-Format konvertieren
erwartetes Ergebnis	Die Aufzeichnung enthält nur die Botschaften, die an das aktive CAN-Interface gesendet wurden.

Tabelle 14 - Testfall T04

Nummer	T04
Titel	Automatisches Starten und Stoppen der Aufzeichnung
Anforderungsbezug	A06
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen
auszuführende Testschritte	1. Datenlogger Konfigurieren (2 CAN-Schnittstellen 500 kbits/s, Betriebsmodus: Permanent, AutoStart: ein) 2. CAN-Botschaften mittels CANoe senden 3. Senden der Botschaften beenden
erwartetes Ergebnis	Die Aufzeichnung startet automatisch, sobald der Datenlogger die erste, mittels CANoe gesendete, Botschaft empfängt. Die Aufzeichnung wird automatisch eine Minute nach der letzten empfangenen Botschaft beendet.

Tabelle 15 - Testfall T05

Nummer	T05
Titel	Automatisches Starten nach manuellem Stoppen
Anforderungsbezug	A06
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen
auszuführende Testschritte	1. Datenlogger Konfigurieren (2 CAN-Schnittstellen 500 kbits/s, Betriebsmodus: Permanent, AutoStart: ein) 2. CAN-Botschaften mittels CANoe senden 3. Aufzeichnung über Konfigurationssoftware Stoppen 4. 1 Minute warten 5. Senden der Botschaften beenden
erwartetes Ergebnis	Die Aufzeichnung startet automatisch, sobald der Datenlogger die erste mittels CANoe gesendete, Botschaft empfängt. Nach dem Stoppen der Aufzeichnung über die Konfigurationssoftware, wird eine neue Aufzeichnung beim Empfang der nächsten Botschaft gestartet. Die Aufzeichnung wird automatisch eine Minute nach der letzten empfangenen Botschaft beendet.

Tabelle 16 - Testfall T06

Nummer	T06
Titel	Betriebsmodus Trigger
Anforderungsbezug	A06
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen.
auszuführende Testschritte	1. Datenlogger Konfigurieren (2 CAN-Schnittstellen 500 kbits/s, Betriebsmodus: Trigger) 2. Beliebige Trigger-Bedingung konfigurieren 3. Konfiguration an Datenlogger senden 4. Aufzeichnung über Konfigurationssoftware starten 5. CAN-Botschaften mittels CANoe senden 6. CAN-Botschaft senden, welche die Trigger-Bedingung erfüllt 7. Aufzeichnung stoppen 8. Aufzeichnung in ASC-Datei konvertieren
erwartetes Ergebnis	Die ASC-Datei enthält einen Eintrag „log trigger event“ nach dem Datensatz, auf dem die konfigurierte Trigger-Bedingung zutrifft. Außerdem enthält die Datei alle CAN-Botschaften, die 2 Sekunden vor und nach dem Auslösen des Triggers empfangen wurden.

Tabelle 17 - Testfall T07

Nummer	T07
Titel	Trigger manuell über Taster auslösen
Anforderungsbezug	A06
Vorbedingungen	Datenlogger an Spannungsversorgung und CANcase angeschlossen.
auszuführende Testschritte	1. Datenlogger Konfigurieren (2 CAN-Schnittstellen 500 kbits/s, Betriebsmodus: Trigger) 2. Keinen Trigger konfigurieren 3. Konfiguration an Datenlogger senden 4. Aufzeichnung über Konfigurationssoftware starten 5. CAN-Botschaften mittels CANoe senden 6. Trigger-Taster betätigen. 7. Aufzeichnung Stoppen 8. Aufzeichnung in ASC-Datei konvertieren
erwartetes Ergebnis	Die ASC-Datei enthält einen Eintrag „log trigger event“. Dieser Eintrag repräsentiert das Betätigen des Tasters. Außerdem enthält die Datei alle CAN-Botschaften, die 2 Sekunden vor und nach dem Auslösen des Triggers empfangen wurden.

Tabelle 18 - Testfall T08

Nummer	T08
Titel	Laden und Speichern von Konfigurationen
Anforderungsbezug	A08
Vorbedingungen	keine
auszuführende Testschritte	1. Mittels Konfigurationssoftware eine beliebe Konfiguration erstellen 2. Konfiguration speichern 3. Konfigurationssoftware beenden 4. Konfigurationssoftware starten 5. Zuvor gespeicherte Konfiguration laden
erwartetes Ergebnis	Die Einstellungen der geladenen Konfiguration entsprechen den zuvor eingestellten Werten.

Tabelle 19 - Testfall T09

Nummer	T09
Titel	Import einer LogDatei in CANoe
Anforderungsbezug	A07
Vorbedingungen	keine
auszuführende Testschritte	1. Eine mit dem Datenlogger aufgezeichnete Logdatei mittels Konfigurationssoftware in das ASC-Format konvertieren 2. Die konvertierte Datei in CANoe importieren
erwartetes Ergebnis	Die Datei wird fehlerfrei in CANoe importiert.

6.3 Testergebnisse

In diesem Abschnitt werden die Ergebnisse der Tests der Betriebs- und Konfigurationssoftware zusammengefasst. Im Test wurden die zuvor unter Kapitel 6.2 definierten Testfälle unter verschiedenen Bedingungen durchgeführt. Es wurden mittels CANoe sowohl hohe als auch niedrige Buslasten für die einzelnen CAN-Schnittstellen simuliert. Die Ergebnisse der einzelnen Testfälle sind in Tabelle 20 zusammengefasst.

Tabelle 20 - Auswertung der Testfälle

Testfall	Ergebnis
T01 Einstellung der Bitrate	i.O.
T02 Betriebsmodus Permanent	Funktion eingeschränkt
T03 CAN-Schnittstelle deaktivieren	i.O.
T04 Automatisches Starten und Stoppen der Aufzeichnung	Funktion eingeschränkt
T05 Automatisches Starten nach manuellem Stoppen	Funktion eingeschränkt
T06 Betriebsmodus Trigger	Funktion eingeschränkt
T07 Trigger manuell über Taster auslösen	i.O.
T08 Laden und Speichern der Konfiguration	i.O.
T09 Import einer Logdatei in CANoe	i.O.

Bei der Durchführung der Testfälle T02, T04, T05, T06 wurde eine Funktionseinschränkung der Betriebssoftware festgestellt. Diese betrifft die Aufzeichnung von CAN-Botschaften bei

hoher Busauslastung. Die hohe Auslastung des CAN-Busses führt zum Überlauf des Puffers für empfangene Botschaften und somit zum Datenverlust. Diese bedeutet, dass durch die Interrupt-Service-Routinen der CAN-Schnittstellen schneller Botschaften im Puffer abgelegt werden als die Betriebssoftware die im Puffer gespeicherten Botschaften verarbeiten kann. Zur Analyse des Problems wurde eine Messung der Verarbeitungszeit pro CAN-Botschaft vorgenommen. Die Messung ergab, dass sich die Verarbeitungszeit der meisten CAN-Botschaften mit 28 µs unterhalb der im Kapitel 4.4.1 maximalen Verarbeitungszeit von 47 µs befindet. Allerdings beträgt die Verarbeitungszeit der ersten Botschaft 158 ms. Dies stellt eine mehr als 3000-fache Überschreitung der maximalen Verarbeitungszeit dar. Der Grund hierfür ist, dass bei der Verarbeitung der ersten Botschaft die Logdatei erstellt wird. Um das Problem zu lösen muss der Vorgang der Datei-Erstellung aus der Verarbeitungsfunktion ausgelagert werden. Möglich wäre hier z.B. das Erstellen der Logdatei während der Initialisierungsphase der Betriebssoftware. Damit der Dateiname der Logdatei trotzdem den Zeitpunkt des Aufzeichnungsstarts enthält, könnte die Datei nach Abschluss der Aufzeichnung entsprechend umbenannt werden. Zusätzlich zur Überschreitung der Verarbeitungszeit bei der ersten Botschaft, zeigen die Messergebnisse weitere Überschreitungen bei der Verarbeitung jeder 39. Botschaft. Eine mögliche Ursache für das zyklische Auftreten einer erhöhten Verarbeitungszeit könnten die in der Betriebssoftware verwendeten Timer sein. Um die genaue Ursache festzustellen sind jedoch weitere Analysen notwendig.

Darüber hinaus traten bei der Durchführung der Testfälle keine weiteren Fehler auf.

7 Fazit und Ausblick

Ziel dieser Diplomarbeit ist es, ein Konzept für die Betriebs- und Konfigurationssoftware eines CAN-Bus-Datenloggers zu erstellen sowie die anschließende Implementierung dieses Konzeptes. Basierend auf einer Analyse bestehender Datenlogger, wurden die notwendigen Anforderungen ermittelt. Diese Anforderungen stellen die Grundlage für das Konzept der Betriebs- und Konfigurationssoftware dar. Die im Anschluss implementierte Software erweitert das b.cube-System um die Funktionalität des Datenloggins für CAN-Busse. Zusätzlich wird durch den modularen Aufbau der Konfigurationssoftware die Basis zur Konfiguration weiterer Komponenten des b.cubes-Systems mittels Windowsapplikation geschaffen. Durch die Verifikation der Software wird nachgewiesen, dass sämtliche definierten Anforderungen erfüllt sind. Lediglich bei einer hohen Busauslastung kommt es zu unvollständigen Daten. Die Ursache dieser Funktionseinschränkung wird aktuell analysiert und in einer späteren Version der Betriebssoftware behoben.

Zusätzlich zu den geforderten Funktionalitäten, wurde die Speicherung der Konfiguration auf dem internen Speicher des Mikrocontrollers realisiert. Durch die Implementierung dieser Funktionalität wird die Benutzerfreundlichkeit wesentlich erhöht. Ein erneutes Konfigurieren des Datenloggers nach Spannungsausfall entfällt somit.

In Summe kann die b.cube-Softwareerweiterung als Erfolg gewertet werden. Im Rahmen dieser Diplomarbeit konnten wichtige Erkenntnisse in Bezug auf das Datenlogging gewonnen werden. Das herausgearbeitete Konzept stellt eine gute Basis für zukünftige Erweiterungen dar. Für die Zukunft ist geplant, die Betriebs- und Konfigurationssoftware um die Funktion zur Aufzeichnung von LIN-Bussen und weiteren Schnittstellen zu erweitern. Weiterhin wird die Implementierung einer Live-Daten-Anzeige in der Konfigurationssoftware beabsichtigt. In dieser sollen die aktuell vom Datenlogger empfangenen Daten angezeigt werden. Voraussetzung hierfür ist die Anpassung der Kommunikationsschnittstelle zwischen Betriebs- und Konfigurationssoftware. Im Zuge dieser Erweiterung kann außerdem die Funktion zum Importieren der Logdateien über diese Schnittstelle implementiert werden. Der Import der aufgezeichneten Daten mittels Konfigurationsschnittstelle ermöglicht den Verbau eines internen Speichermediums.

Geschlossen wird diese Diplomarbeit mit einem kurzen Zitat: „Es sieht so aus, als würden wir den größten Wandel erleben, den das Auto in seiner über 120-jährigen Geschichte gesehen hat“ so Ferdinand Dudenhöffer [30].

Durch die Elektromobilität und den Einsatz der CarIT wird es in den kommenden Jahren bei der Entwicklung von Fahrzeugen zu einem steigenden Bedarf an Datenanalysemöglichkeiten kommen [30]. Mit b.cube macht Bertrandt einen ersten Schritt, um diesen Bedarf zukünftig bedienen zu können.

Literatur

[1] Burton, Simon: Entwicklungsprozess für funktional sichere Steuergeräte, http://www.elektroniknet.de/automotive/tools/artikel/121095/, verfügbar am 22.01.2016

[2] Zimmermann, Werner; Schmidgall, Ralf: Bussysteme in der Fahrzeugtechnik, Wiesbaden, Springer-Vieweg, 2014

[3] Bertrandt Ingenieurbüro GmbH Tappenbeck: b.cube Produktkonzept

[4] G.i.N. mbH: Flyer GL1000, http://gin.de/files/3949/flyergl1000de.pdf, verfügbar am 15.12.2015

[5] Telemotiv AG: blue PiraT Mini Handbuch, https://sc.telemotive.de/4/uploads/media/blue_PiraT_Mini_Benutzerhandbuch.pdf, verfügbar am 15.12.2016

[6] Vector Informatik GmbH, Produktinformation GL-Logger-Familie, http://vector.com/portal/medien/cmc/info/GL_Logger_ProductInformation_DE.pdf, verfügbar am 15.12.2016

[7] Bertrandt AG: Grundlagen CAN/ CANoe / CANalyzer, interne Schulungsunterlagen

[8] ARM Ltd.: EA LPC4088 QuickStart Board, https://developer.mbed.org/platforms/EA-LPC4088/, verfügbar am 18.12.2015

[9] ARM Ltd.: The ARM mbed IoT Device Platform, http://www.mbed.org, 18.12.2015

[10] Vector Informatik GmbH: Produktinformation CANoe, http://vector.com/portal/medien/cmc/info/CANoe_ProductInformation_DE.pdf, verfügbar am 05.01.2016

[11] Idealo.de: Preisvergleich SDHC-Karten 16 GB, http://www.idealo.de/preisvergleich/ProductCategory/4734F890746-1294945.html?q=sd-karte,verfügbar am 26.03.2016

[12] Idealo.de: Preisvergleich USB-Stick 16 GB, http://www.idealo.de/preisvergleich/ProductCategory/4312F2105766.html?q=usb-stick,verfügbar am 26.03.2016

[13] Embedded Artists: MCIFileSystem, https://developer.mbed.org/users/embeddedartists/code/EALib/file/e1e36493f347/MCIFileSystem.h, verfügbar am 17.12.2015

[14] Okamoto, Norimase: LPC4088-USBHost, https://developer.mbed.org/users/va009039/code/LPC4088-USBHost/, verfügbar am 17.12.2015

[15] mbed.org: CAN, https://developer.mbed.org/handbook/CAN, verfügbar am 20.03.2016

[16] Mikrocontroller.net: FIFO, http://www.mikrocontroller.net/articles/FIFO, 20.03.2016

[17] mbed.org: Timer, https://developer.mbed.org/handbook/Timer, 20.03.2016

[18] mbed.org: Ticker, https://developer.mbed.org/handbook/Ticker, 20.03.2016

[19] mbed.org: Serial, https://developer.mbed.org/handbook/Serial, 20.03.2016

[20] Wallentowitz, Henning; Reif, Konrad: Handbuch Kraftfahrzeugelektronik, Wiesbaden, Vieweg, 2006

[21] cppreference.com: std::deque , http://de.cppreference.com/w/cpp/container/deque, verfügbar am 10.02.2016

[22] Embedded Artists: EALib, https://developer.mbed.org/users/embeddedartists/code/EALib/, verfügbar am 10.02.2016

[23] Schwichtenberg, Holger: .NET-Oberflächen mit Windows Forms oder WPF?, http://www.heise.de/developer/artikel/NET-Oberflaechen-mit-Windows-Forms-oder-WPF-227252.html, verfügbar am 01.03.2016

[24] Kühnel, Andreas: Visual C# 2012, http://openbook.rheinwerk-verlag.de/visual_csharp_2012/, verfügbar am 01.03.2016

[25] Mbed.org: mbed sdk coding style, https://developer.mbed.org/teams/SDK-Development/wiki/mbed-sdk-coding-style, verfügbar am 20.03.2016

[26] Stack.nl: Doxygen, http://www.stack.nl/~dimitri/doxygen/, verfügbar am 20.03.2016

[27] Microsoft: XML-Dokumentationskommentare, https://msdn.microsoft.com/de-de/library/b2s063f7.aspx, verfügbar am 23.03.2016

[28] EWSoftware: Sandcastle, https://github.com/EWSoftware/SHFB, verfügbar am 23.03.2016

[29] Vector Informatik GmbH: Handbuch CANcaseXL, https://vector.com/portal/medien/cmc/manuals/CANcaseXL_Manual_DE.pdf, verfügbar am15.03.2016

[30] Audimax: Die Zukunft der Automobilbranche, https://www.audimax.de/ingenieur/automobilbranche/, verfügbar am 28.03.2016

Anlagen

Teil 1 ..... A-I

Anlagen, Teil 1

Inhalt der beigefügten CD

Ordner	Inhalt
\01_Diplomarbeit	Diplomarbeit im Portable Document Format (PDF)
\02_Betriebssoftware\01_Quellcode	Der Quellcode der Betriebssoftware
\02_Betriebssoftware\02_Dokumentation	Mittels Doxygen erstelle Dokumentation der Betriebssoftware
\03_Konfigurationssoftware\01_Quellcode	Quellcode der Konfigurationssoftware
\03_Konfigurationssoftware\02_Dokumentation	Mittels Sandcaste erstellte Dokumentation der Konfigurationssoftware
\03_Konfigurationssoftware\03_ausfuehrbares_Programm	Ausführbare Konfigurationssoftware (exe)

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Leiferde, den 29.03.2016

Andreas Schmidt